Detecting Live Forensic Memory Acquisition using Distinct Native Attribute Fingerprinting

Dinesh Mothi

School of Computer, Engineering, and Media

De Montfort University

A thesis submitted in partial fulfillment for the degree of

Doctor of Philosophy

De Montfort University December 2020

Declaration

I hereby declare that the materials of this submission have not previously been published for a degree, or diploma at any other university or institution. All materials submitted for assessment are from my own research, except the reference work in any format by other authors, which are properly acknowledged in the content. Part of the research work presented in this submission has been published, or has been submitted for publication in the following paper:

Mothi, D., Janicke, H. and Wagner, I., 2020. A novel principle to validate digital forensic models. Forensic Science International: Digital Investigation, p.200904.

Dinesh Mothi

December 2020

Acknowledgements

First and foremost I would like to thank my supervisors Dr Isabel Wagner and Dr Fabio Caraffini for providing their guidance and support throughout what has been a challenging and an amazing journey.

I would also thank the Doctoral College for their time, support, and answering any queries that I had during the course of my study.

I would love to thank my family for all their support and motivation without which my research journey would not have been possible. My partner for all her time, support, understanding, and all the help I got during difficult times, especially during the lockdown. Thank you for making this journey a success.

Also thanks to all my friends for your support, kindness, time, and enthusiasm.

Abstract

This research investigates the anti-forensic aspects of live memory acquisition. In 2009, the anti-forensic tool Detect and Eliminate Computer Assisted Forensics (DECAF) was developed to defeat one of the forensic tools, which is well known in the law enforcement community, Computer Online Forensic Evidence Extrator (COFEE). DECAF uses signature based detection method to detect forensic tools, and then performs anti-forensic routines such as modifying evidence, disabling forensic software tools, shutting down the machine to avoid evidence collection, to name a few. The findings in the literature show that the signature based method and various anti-forensic methods have shortcomings. This led to review the application of artificial intelligence (AI) techniques, specifically machine learning algorithms, to the domain of anti-forensics. And, also that AI is not being applied to detect the live forensic acquisition process. This is the knowledge gap this study has identified and addressed to the extent that AI techniques can be applied to detect forensic memory acquisition on a Windows 10, 64-bit machine.

The method that was adopted to address the knowledge gap was by formulating a hypothesis, that the memory (M), input and output (I/O), and central processing unit (C) parameters exhibit a distinct variation in MIOC pattern, known as distinctive native attribute (DNA) pattern or fingerprint, whilst memory is being acquired from the machine by a forensic tool. If these unique DNA patterns can be identified, then memory acquisition can be detected. To support the hypothesis of this study, an experiment was conducted to gather MIOC parameters, and machine learning (ML) algorithms were used to identify the DNA patterns.

The results show that, adaptive (ADA) boost classifier has the least performance with a high detection error rate (Δ_r) of 73.4 percent and 89 percent on the memory and CPU dataset respectively. Whereas, linear discriminant analysis (LDA) classifier has the highest Δ_r of 78 percent on the I/O dataset. Random forest (RF) classifier has the least detection rate of less than 5 percent on the three datasets. To improve the performance of the ML algorithms the individual memory, I/O, and CPU datasets was integrated into the single MIOC dataset. This resulted in decreasing Δ_r of SVM, LDA, and ADA boost by 32 percent, 17.3 percent, and ADA 13 percent respectively.

To further support the hypothesis, the MIOC dataset was transformed to images by using the concept of gramian angular field (GAF). Then, the DNA patterns were detected using the 3-layered convolution neural network (3L-CNN) model. The results show that the model detects DNA patterns from the I/O dataset with an accuracy, precision, and recall over 99 percent. Whereas, the model underforms on the CPU dataset with an accuracy of 64.94 percent with precision and recall of 69.50 and 44.20 percent respectively. The significance of DNA fingerprinting detection method is that, it not only shows that AI techniques can be applied to the anti-forensic domain but also highlights that forensic memory acquisition can be vulnerable to the DNA fingerprinting method. This implies, if memory acquisition could be detected using DNA fingerprinting, the process of live memory acquisition is defeated. Therefore, the investigator will not have crucial evidence to work with that could be found only in the memory.

Another novel contribution this study makes is by proposing a mathematical formalism by which a digital forensic model (DFM) can be validated by counteracting the influence anti-forensic effects on various phases of the digital forensic process. Future work should focus on addressing live forensic acquisition vulnerabilities.

Table of contents

List of figures

List of tables

Glossary

ACPO Association of Chief Police Officers. The good practice guidelines for digital forensics..

AF Anti-Forensics. The procedure used to defeat the process of digital forensics..

BIOS Basic Input Output System. A program used by the microprocessor to start the system when the machine is turned on.

BSOD Blue Screen of Death. A kernel mechanism of the Windows OS where an unwarranted event has occurred.

CNN Convolution Neural Network. A deep neural algorithm which is based on the mathematical principle of convolution.

COFEE Computer Online Forensic Evidence Extractor. A live forensic tool available to the law enforcement community.

CPU Central Processing Unit. A computer unit responsible for performing robust calculations and operations to get tasks completed on a machine.

DECAF Detect and Eliminate Computer-Assisted Forensics. An anti-forensic tool used to defeat the live forensic process.

DF Digital Forensics. A branch of forensic science concerning the investigation of digital devices..

DFM Digital Forensic Model. An iterative process flow to illustrate the process of DF.

DL Deep Learning. A subset of AI that implements neural networks by mimicking the human brain.

DMA Direct Memory Access. A feature of computer system allowing a hardware subsystem to access memory without processor's involvement.

DMA Virtual Address Descriptor. A self-balancing binary tree stored in the enivornment process of memory.

DNA Distinct Native Attribute. The features which are unique and local to a variable.

EFI Extensible Firmware Interface. An interface between the OS and the firmware.

epochs Indicates the number of time a ML or DL algorithm has passed a dataset.

EPROCESS Kernel datastructure that represents a process object.

FPR False Positive Rate. Proportion of negatives truly identified.

GAF Gramian Angular Field. A mathematical concept to retain the temoporal dependencies in time-series data.

GSF Gramian Summation Field. A Property of GAF.

I/O Input/Output. In this research, it is the number of read/write operation during memory and non-memory acquisition..

IDT Interrupt Descriptor Table. A datastructure for 32-bit OS architecture that implements a list of interrupts.

ISO 17025 International Standard Organisation 17025. A certification involving testing and calibration of laboratories.

ISO/IEC 27037 Good practice guidelines for indentification, acquisition, and preservation of evidence.

ISO/IEC 27043:2015 Good practice guidelines for processes and principles concerning incident investigation.

memory Also known as the Random Access Memory which stores data until the loss of power.

MIOC Memory, I/O, and CPU parameters that are acquired in this study to argue and support the phenomenon of forensic memory detection.

ML Machine Learning. A branch of AI that gives a machine the ability to learn without user intervention.

precision The proportion of actual instances to the retrieved instances.

RAM Random Access Memory. The volatile memory loses its contents when the power is turned off from the machine..

recall The percentage of total number of actual instances retrieved.

ReLU Rectified Linear Unit. Also known as rectifier, it is a linear fucntion that outputs the input if it is positive. Outputs zero otherwise.

ROC Receiver Operating Characteristic. A graphical plot of true positive rate and false positive rate.

SOP Standard Operating Procedure. A set of instructions by an organisation to execute DF procedures.

SSDT Internal dispatch table for Windows OS containing relative offsets for kernel routines.

TLB Translation Lookaside Buffer. A high speed memory cache for page table entries.

TPR True Positive Rate. Proportion of positives truly identified.

Chapter 1

Introduction

1.1 Introduction

This research focuses on the anti-forensics of the live forensic acquisition process. Digital Investigation is a process through which digital evidence is examined in an orderly fashion to infer what had happened previously. The forensic process is guided by the digital forensic models or good practice guidelines such as Association of Chief Police Officers (ACPO), (Williams, 2012). Anti-forensics (AF) on the other hand, is the process that aims to defeat digital forensic process. This research proposes a method to detect forensic memory acquisition and a novel principle to validate digital forensic models by counteracting anti-forensic attacks.

1.2 Background

Forensics is the application of scientific knowledge to the legal domain, and the application of a specific scientific area that concerns various scientific disciplines to legal, civil, and judiciary matters is the domain of forensic science. Digital forensic science is defined as the use of scientifically proven methods to identify digital evidence to reconstruct events to assist

an investigation, (Carrier, 2002). During the course of an investigation, when one encounters electronic devices such as computers, they might find it to be powered on and functioning. Traditionally, the investigators on the field will pull the power chord off the computer's cabinet to shutdown the machine. The advantage of this method is that the operating system will not have the chance to flush any data back onto the hard drive, and this would mean that the integrity of the evidence has been preserved at a point in time. However, the volatile data that is stored in the physical memory, that is, Random Access Memory (RAM), would be lost if one decides to perform traditional digital forensic methodology to gather evidence from computers during an investigation (Hay et al., 2009).

The shortcomings of the traditional approach are the situations where it would be impractical to image a hard drive given its large volume which could extend to more than hundreds and thousands of terabytes (Quick and Choo, 2014). It may not be possible to take a computer system down due to business-critical operations such as a server in a financial institution. Cases where drives are encrypted also pose a threat to an investigation if the encrypted keys are not obtained prior to imaging the hard drive. Also, criminals who are aware of the traditional methodology develop malicious software so that they are run only in the physical memory, and do not leave any traces behind on the hard disk (Reyes et al., 2007). To overcome the shortcomings of traditional forensics, prior to shutting down the system, investigators would gather volatile data, or may image the hard drive whilst the machine is powered on. As for the former case, the investigators would install their known-good tools on a USB drive or a CD/DVD. These tools then gather specific information from the physical memory and certain areas of the hard drive depending on the case in hand (Adelstein, 2006). As the field of digital forensics was progressing during the last two decades, researchers began to look into ways by which one could defeat the process of digital forensic investigation. They then termed the methods, techniques, and tools that defeats the process of an investigation as anti-forensics (Garfinkel, 2006). Anti-forensic methodologies can delay the process of

an investigation which is known as trial obfuscation, and prevents an investigator to have access to the evidence. The most popular scenarios include encryption of certain files within the storage media or encrypting the medium itself. It can also manipulate the evidence on the fly especially when the investigator gathers evidence from the physical memory to later investigate the memory dump in the lab (Harris, 2006).

The manipulation of evidence during a digital forensic phase would result in reaching a wrong conclusion about a given hypothesis in the analysis phase. This results in gathering incomplete evidence;thereby, leaving the investigator to work without any vital evidence. During live forensic investigations, this circumstance may occur if an investigator does not take proper precautions during the acquisition process.

1.3 Motivation

In 2009, two security researchers developed an anti-forensic tool known as DECAF (Detect and Eliminate Computer-Assisted Forensics) to detect the live forensic tool COFEE (Computer Online Forensic Evidence Extractor) which is distributed among the law enforcement community. DECAF is an anti-forensic tool that had claimed to detect forensic tools such as COFFEE, FTK Imager, EnCase, and perform anti-forensic (AF) operations on the machine. Later in 2009, the authors of DECAF made an official statement that the AF tool is an hoax, and was a publicity stunt to raise awareness in the forensic community regarding the security issues in the DF domain (Mansfield-Devine, 2010).

The existence of this anti-forensic tool could possibly pose a threat to live forensic investigations. However, if the nature of the tool could be studied, then risks to the DF investigation process can be minimized. To the best of my knowledge, it appears DECAF tool is not available on the internet, so it was not possible to conduct experiments to test memory forensic acquisition tools against it. Also, another aspect of DECAF that has guided this research work was the method by which DECAF detects forensic acquisition tools. It detected live forensic

tools by their file signature that included name of the forensic tools (Lim et al., 2012). After implementing a proof-of-concept of DECAF signature based method in section 2.4, it appears DECAF executes in user-mode. From 2.3, it is evident that various anti-forensic attacks mostly work in the kernel of Windows-32 bit machines. Implementing those procedures on 64-bit Windows 10 machines would cause a blue screen of death (BSOD). Therefore, in this research a method was proposed to detect forensic memory acquisition using artificial intelligence techniques on 64-bit Windows 10 machines in user-mode.

1.4 Knowledge Gap and Problem

This thesis reports an investigation on anti-forensic techniques affecting various phases in the digital forensic process, specifically focusing on the acquisition phase. In section 2.5, the research identifies that artificial intelligence (AI) techniques are being applied to assist the digital investigation process, for example, to detect malware present in a memory forensic image. However, the findings in the literature show that artificial intelligence is not being applied to the anti-forensic domain. To this extent, the aim of this study is:

Aim 1: To accuractely detect memory acquisition on Windows 10 machine using artificial intelligence techniques

To meet Aim 1 and address the aforemention knowledge gap, the first central research question (RQ) was posed:

RQ1: How can forensic memory acquisition be accurately detected by artificial intelligence techniques? In this research, memory is acquired directly from the computer using a USB device. Memory is not acquired remotely over a network. The corresponding research objectives for RQ1 are:

1. Differentiate between forensic memory acquisition and non-acquisition using machine learning classifiers and evaluate their accuracy, precision, and recall metric as it can

help identifying which machine learning classifier performs well in distinguishing between memory acquisition and non-acquisition.

2. Cross-validate the results (evaluation metrics) of the machine learning classifiers with learning curves and bias-variance trade-off. The reason for employing various strategies to verify the results is to enhance and increase the confidence in the findings, (Henry, 2018). It also helps in determining the reason why a model does not perform well. For example, a learning curve shows whether a classifier fits the data well or not, and this may be due to lower precision or recall which in turn could affect the model's accuracy.

3. Encode the samples in the dataset into images and classify them using convolution neural network (CNN). A CNN will be used as it is well known for classifying images (Tsai et al., 2019). And, determine model's performance by comparing accuracy, precision, recall, false positives, and false negatives.

The findings in sections 2.1 and 2.3 shows that the major phases of the digital forensic (DF) process such as acquisition, examination, analysis, and reporting are affected by anti-forensic techniques. To address this problem, the following aim and the second research question was posed:

Aim 2: To validate digital forensic models in the presence of anti-forensic attacks

RQ2: How can digital forensic models be validated when they are affected by anti-forensic techniques?

To answer the RQ2 and meet Aim 2, the corresponding research objective is:

Formalize a generic framework to validate digital forensic models by counteracting anti-forensic techniques.

1.5 Contribution to Knowledge

The contribution of this thesis fills a knowledge gap and addresses a problem in the field of anti-forensics.This research proposes a method by which memory forensic acquisition can be detected. This is facilitated by designing an experiment to gather memory, I/O, and CPU (MIOC) parameters and then detecting memory forensic acquisition and non-acquisition activity with the help of various machine learning (ML classifiers that are evaluated in chapter 4.

The second contribution is transforming the MIOC dataset into images by using the Gramian Summation Field (GSF) property of the Gramian Angular Field (GAF) (Tsai et al., 2019). The memory acquisition and non-acquisition distinct native attribute (DNA) patterns were then detected using 3-Layered Convolutional Neural Network (3-L CNN) model which is evaluated in chapter 5.

The significance of these contributions are that it shows the acquisition phase of the live forensic process is vulnerable to the DNA pattern detection technique that was proposed in this research. The Association of Police Officers (ACPO)-Good Practice Guidelines for Digital Forensics (Williams, 2012) is an important reference document when conducting digital forensic investigations and is referred by the law enforcement agencies in the United Kingdom (UK). In Williams (2012, Pg. 26), the steps required to perform live forensics is documented, and if an investigator performs those steps to acquire memory, then memory acquisition can be detected using the methods proposed in this research work. Also, the machine learning algorithms and the 3L-CNN model were implemented in Python. If this memory detection software is executed on a windows 64-bit machine, it runs in the user-mode just like any other process on the machine and it will not show any suspicious behaviour unlike malware processes to the forensic investigator. In an event, even if the investigator tries to analyse this particular memory detection executable they would have to apply malware analysis techniques. This will either obfuscate the investigation process or will trick the

investigator into reaching wrong conclusions, because analysing the MIOC-DNA pattern detection (or fingerprint) method proposed in this study has not been researched by malware researchers to date. Therefore, the state-of-art malware analysis techniques (Or-Meir et al., 2019) will fail to analyse the memory detection procedure proposed in this research.
Another contribution this study makes is that of a novel mathematical formalism to validate digital forensic models (DFMs) if its under the influence of an anti-forensic attack. The significance of this contribution is that an investigator would be able to validate their DFM if an anti-forensic techniques is affecting a particular phase in their digital forensic model. By doing so, they also will address the issue of validating their forensic tools for anti-forensic techniques as this is on a lower level when compared to DFM as seen in chapter 6. Therefore, the novel principle proposed in this research to validate digital forensic models would be serve as a guide for investigators to validate their process and tools in order to comply with the ISO 17025 standard in the UK that validates forensic tools (Marshall and Paige, 2018).

1.6 Organisation of Thesis

Chapter 2 reviews various digital forensic models, anti-forensic techniques to ascertain what phases of the digital forensic process is affected by anti-forensic techniques. It also reviews various ML algorithms applied to the digital forensic and cybersecurity domain.
Chapter 3 discusses the research methods and proposes the research method for this study. It also justifies the purpose of designing the experiments, the collecting and pre-processing the data, and various data analysis techniques.
Chapter 4 evaluates and compares various machine learning classifiers on the MIOC dataset. The ML classifiers were evaluated based on their confusion matrix parameters which are accuracy, precision, and recall. These results were cross compared with receiver operating characteristics (ROC) curve, Precision and Recall (P-R)curve, and learning curve to determine if any model exhibited high bias or variance, that is, whether the model has overfit, underfit,

or had fit the data well.

Chapter 5 to further support the results of chapter 4, another method to detect forensic memory acquisition is proposed in this chapter. MIOC parameters are transformed to images by using the Gramian Summation Field (GSF) property of the Gramian Angular Field (GAF), and then the DNA patterns are identified by the 3 Layer-Convolutional Neural Network to differentiate between memory acquisition and non-acquisition.

Chapter 6 proposes a novel mathematical formalism to validate digital forensic models by counteracting anti-forensic techniques. A paper was submitted in the well renowned journal publication in digital forensics – Forensic Science International: Digital Forensics.

Chapter 7 discusses how the research questions were answered and the objectives met. It also summarises the contribution in this research and critically evaluates it with the state-of-art research.

Chapter 2

Literature Review

The purpose of this chapter is to critically review digital forensic models, anti-forensic techniques, and applications of AI to digital forensics. Firstly, the digital forensic models are reviewed to see whether anti-forensic techniques influence a phase in the digital forensic process. This guided the research to ascertain what phases in the forensic process are affected by anti-forensic techniques and what procedures can be implemented to counteract those affects. Secondly, various acquisition tools are reviewed in section 2.3 to find out if there are any tools resistant or vulnerable to anti-forensic attacks. This is necessary because it gives an insight into what measures the forensic tools adopt to address anti-forensic attacks, or it can show the shortcoming or weakness in the forensic tools when it comes to countermeasuring anti-forensic methods.

Thirdly, applications of AI to digital forensics and anti-forensics are reviewed to find how AI techniques aid digital investigation, and what AI techniques work against it. Although, application of AI to anti-forensics is still in its infancy, this area is reviewed to explore if there are any AI techniques in the literature that has a negative impact on the digital forensic process. Fourthly, a signature based detection method is implemented as a proof-of concept to demonstrate how a forensic tool can be detected due to the underlying weakness in the ACPO guidelines, Williams (2012, Pg. 26), identified in this research, and the shortcoming of

the method is ascertained. Finally, the knowledge gap and the research problem is identified and research questions are posed to fill the gap and address the problem.

2.1 Review of Digital Forensic Models

The generic computer forensic investigative model (GCFIM) developed by (Yusoff et al., 2011) groups common computer forensic phases from previous DFMs. It consists of five phases: pre-process, acquisition, analysis, presentation, and post-process. GCFIM model can serve as a high-level computer forensic investigation model and also could assist in creating new computer forensic investigative methodologies. Similarly, by comparing and expanding over previous computer forensic models, the Systematic Digital Forensic Investigation Model (SRDFIM) was proposed by (Agarwal et al., 2011), which consists of eleven phases namely preparation, securing the scene, survey and recognition, documenting the scene, communication shielding, evidence collection, preservation, examination, analysis, presentation and, result and review. This model helps in reconstructing events by realizing certain properties such as individuality, repeatability, reliability, performance, testability, scalability, quality and standards in analysis pertaining to computer frauds and cyber crimes. Whereas, (Soltani and Seno, 2019) design their event reconstruction model using formal methods such as temporal logic which is an automatic verification technique, and is evaluated on the file allocation table (FAT) file system. Eleven digital forensic models were assessed and evaluated by (Montasari, 2016) using the five criteria that is set in the Daubert Standard, a standard that is used to accept scientific evidence in the United States. None of the eleven models could satisfy all the conditions of the Daubert Standard as argued by (Meyers and Rogers, 2005). It was deduced that, (Ciardhuáin et al., 2009) and (Rogers et al., 2006) took the most scientific approach to develop their digital forensic models. Argument is also made that the DFMs did not include the full scope of the investigation, and only concentrate on a few aspects of the digital forensic investigation. The DFMs were based on personal experience and no model

can be considered as a standard one, and the error rate could not be calculated. The existing models are not flexible, that is, they are not generalised and cannot be applied to different domains of digital forensics (Montasari, 2016).

During the last decade specific DFMs developed by (Hitchcock et al., 2016), (Kaur et al., 2018), (Ali et al., 2017), (Zia et al., 2017), and (Lutui, 2016) focus on certain areas of digital forensics such as digital triage, network forensics, mobile forensics, and Internet of Things (IoT) forensics respectively. To increase the effectiveness and efficiency of IoT investigations (Oriwoh et al., 2013) propose 1-2-3 Zones in conjunction with Next-Best-Thing Triage (NBT) model where necessary by maximising the utilisation of time to ensure relevant evidence is identified and acquired. The NBT model and 1-2-3 zones were adopted by (Harbawi and Varol, 2017) into their digital forensic procedure flowchart and argue that their theoretical framework for IoT investigations is improved and copes with evidence acquisition issues, and to analyse potential digital evidence in an IoT ecosystem, (Kebande et al., 2018) proposes an integrated digital forensic framework for IOT devices.

Various researchers also adopted International Standard Organization (ISO) standards into their digital forensic framework to either enhance or hope to standardise their framework. To improve speed and quality of investigations (Kao and Wu, 2016) propose a digital triage forensics framework of windows malware forensic toolkit based on ISO/IEC 27037, which provides guidelines with respect to common scenarios encountered during the digital evidence handling process and assists organisations in their methods and procedures, and also facilitates the exchange of digital evidence between various jurisdictions. A forensic-by-design framework to enhance the framework for digital forensic investigations pertaining to cyber-physical cloud system that aids organisations to recover from a cyber-physical attack based on ISO 27043:2015 was developed by (Ab Rahman et al., 2016) and this provides guidelines on various investigation scenarios involving digital evidence. Similarly, Kigwana et al. (2017) propose a digital forensic investigative framework which is also based

on ISO/IEC 27043:2015 to develop a standard eGov forensic investigation procedure, and Karie et al. (2019) argue that key factors such as blockchain should be added to ISO/IEC 27043:2015 to support standardised digital forensic report generation process.

Only two frameworks, (Rekhis and Boudriga, 2012a) and (Rani and Kumari, 2017), were found in the literature that consider to detect anti-forensic attacks. A digital forensic process consisting of five phases was proposed by (Rekhis and Boudriga, 2012a) that takes detection of anti-forensic attacks into consideration. However, their framework can detect anti-forensic attacks only in the analysis phase of the digital forensic process. Their framework is supported by several complex propositions to develop hierarchical visibility theory to detect anti-forensic attacks as proposed in (Rekhis and Boudriga, 2012b). A case study is also provided explaining how their propositions are applicable to a case where an administrative account has been compromised.

Another framework that proposes to detect anti-forensics attack in a cloud environment is proposed by (Rani and Kumari, 2017). Their framework is based on (Rekhis and Boudriga, 2012a) framework to detect anti-forensic attacks. It consists of six phases, and the authors only mention to use attack graphs in order to detect the AF attacks, but do not explicitly show how graph theory or attack graphs can be used to detect AF attacks in a cloud investigative environment. In their framework also, the anti-forensic attacks are proposed to be detected in the analysis phase, but by reviewing the literature, the research has identified that anti-forensic attacks can affect not only the analysis phase but also various important phases in a digital forensic framework such as collection, examination, and reporting as shown in Table 6.1.

The review of the digital forensic models informs this study that various versions of digital forensic models are being proposed in the literature by adding or deleting certain phases in the previous models. And, anti-forensic affects have not been taken into account whilst developing or enhancing digital forensic models. As the objective of this research is to

investigate what phases of the digital forensic process is vulnerable to anti-forensic attacks, specifically, the aquisition phase of the live forensic process. The following section reviews various acquistion tools to ascertain their vulnerability or resistance to anti-forensic attacks.

2.2 Vulnerabilities in Memory Acquisition Tools

Volatile memory evidence is critical to a digital investigation, and is highly recommended to be included in the digital investigation process that is used to analyse a crime scene. In response to the increase in volume of evidence, encryption and with the main focus being on the survey phase of the digital investigation process, Volatools, was developed (Walters and Petroni, 2007). Another way to tackle live forensic issues is by a 2-take approach proposed by (Law et al., 2009) for acquiring physical memory images. In this approach, two consecutive snapshots of the memory are taken, and this is done to ascertain the confidence level is higher with respect to presenting the evidence in the court of law. By this methodology the integrity of the memory images can be verified by comparing their preserved memory area for any changes in the data that might have taken place. Various memory acquisition tools were compared with each other with regards to how much space they occupied in the memory, and how much percentage change each tool induces whilst acquiring the evidence from the physical memory. This in turn was compared to the methodology of taking a single memory snapshot by (Law et al., 2009), and they argue that this procedure, of single-snapshot memory acquisition, is unreliable and unauthentic because it could be challenged in the court of law with regards to integrity of evidence not being verified.

By comparing various volatile memory acquisition tools with respect to their impact on memory in terms of memory footprint their potential impact on the machine under examination can be analysed. Also, tools used to obtain relevant data from memory can be tested to ascertain the impact on the file system, Registry and DLL usage. This criteria used for assessing the volatile memory tools should provide a guide for digital forensic examiners

whilst investigating Windows OS based computers (McDown et al., 2016). Whereas, the system image and volatile memory data to ascertain the exact state of the computer system in question, and the sequence of events that led to its attack is a challenging task in live forensics. In addition to the advantages of the static analysis methodology has, it also consists of data that is present in the volatile memory such as process hidden in the memory, for example, a visible rootkit. Also, virtual environment facilitates the process of booting the image for interactive investigation purposes, and the advantage of this method is that the original image is, preserved and in pristine condition, and the analysis is repeatable. The evidence gathered during this process would be admissible in the court of law (Mrdovic et al., 2009).

To address integrity issues in live forensics (Vömel and Freiling, 2012) have defined correctness, atomicity, and integrity as the three factors for evaluating the quality of a forensic memory snapshot. The actual values of addressable memory regions are contained in a correct snapshot at a specific point of time that were stored in those regions. Whereas, an atomic snapshot does not include any signs of concurrent system activity. The stable values of memory regions can be estimated by the level of integrity that remain in the course of the imaging operation. It can specifically measure the impact of certain acquisition methodologies and differenciate against more invasive software based methods that loads modules into the memory (kernel); therefore, destroying important pieces of evidence.

Also, various unique factors like correctness, atomicity, and integrity of the acquired memory can be measured that establishes the quality of memory images to provide a platform to evaluate open-source memory forensic acquisition software tools such as win32dd (Suiche, 2013), WinPMEM which is now a part of Rekall (Rek, 2019), and ManTech Memory DD (MDD)(Man, 2015) can be done by using a white-box testing methodology. The memory sizes can range from 512 MB to 20 GB, and 270 snapshots from different systems were analysed in an idle state. It was revealed that a few software tools were not able to acquire an image of the complete physical address space, and also these incomplete memory images

produced mismatched data offsets. The analysis of these memory images might lead to false conclusions at a later stage in the investigations, in order to overcome this fault different utilities were patched. An interesting observation made here was that the atomicity decreased as the memory sizes increased, and based on this observation it was argued that larger the memory size longer the acquisition process, and it would be difficult to keep the memory images free from inconsistencies due to concurrent activity. Whereas, it was observed that the integrity of the memory images increased with system that had a constant load because less areas of memory were subject to change proportionally (Vömel and Stüttgen, 2013). Whereas, (Gruhn and Freiling, 2016) present a black box methodology to evaluate atomicity and integrity of memory acquisition methods with respect to the correctness of a memory acquisition procedure, and focus on the soundness of memory acquisition tools and methods. Several memory acquisition methods were characterised and twelve memory acquisition tools were evaluated upon a windows 8 64-bit operating system. It was found that, the user-mode acquisition software such as ProcDump and Windows Task Manager, memimage, msram-dump, virtualisation using virtualbox, and emulation using QEMU provide perfect atomicity and integrity of complete RAM snapshot. Whereas, Kernel level memory acquisition tools such as FTK Imager, DumpIt, win64dd, and WinPmem show concurrent system activity which reduces its atomicity, and integrity is reduced because the acquisition tools might overwrite some part of the memory space. DMA attack using inception IEEE 1394 showed the least atomicity, but its integrity is inferior when compared to any of the aforementioned methods.

Six windows memory acquisition tools such as Moonsols DumpIt, AccessData FTK Imager 2016, Winpmem, Belkasoft RAM Capture, Mandiant's Memoryze (2016), and Magnet RAM Capture, were comapred on a Windows 32-bit Ultimate machine (Ahmed and Aslam, 2015). These six memory acquisition tools were compared with each other using three memory analysis tools namely: Volatility Framework 2.4, Windows SCOPE Ultimate,

and Mandiant's Redline. The data sets used for these comparisons were Processes, Driver Modules, Device Tree, SSDT, IDT, Drivers, Registry Keys, and Network Sockets. Two scenarios, Scenario I and Scenario II, were framed basing on which the physical memory was acquired. The former consisted of no anti-debugging tool, and the latter consisted of an anti-debugging tool named nProtect GameGuard. The results showed that all the six memory acquisition tools performed normally, and the data was accurately captured from the RAM and was analysable using the three memory analysis tools. Whereas in the second scenario it was deduced that, FTK Imager and Magnet RAM capture, could not accurately acquire the physical memory because they could not be analysed by the three memory acquisition tools. Various firmware rootkits such as Basic Input Output Sysytem (BIOS) rookits, Extensible Firmware Interface (EFI) rootkits, Advanced Configuration and Power Interface (ACPI) rootkits leave traces in the physical memory. Firmware rootkits are very dangerous because they can subvert the base of the machine, and the associated memory traces are highly likely to be overlooked during an investigation. It can be shown that the software memory acquisition tools such as memoryze, FTK Imager , Moonsols DumpIt, Windows Memory Reader, LiMe, WinPmem, and Pmem fail to acquire all parts of the physical memory, especially those parts of the memory address spaces that contain the firmware data. This drawback has been eliminated by inventing the memory acquisition tools, WinPmem and Pmem. Thus, facilitating the memory analysis of the firmware rookits that leave their traces in the physical memory, (Stüttgen and Cohen, 2013). An in-depth analysis of memory forensic acquisition tools was presented by (McDown et al., 2016). An observation was made regarding Windows Memory Reader and Belkasoft's Live Ram Capturer–these tools left the least footprints when loaded in memory. Whereas, ProDiscoverer and FTK Imager perform poor in memory usage, processing time, DLL usage, and unwanted artefacts were introduced into the system. It was also found that Belkasoft's Live Ram Capturer is the fastest to obtain an image of the memory, and ProDiscoverer is the slowest to image memory.

2.3 Anti-Forensic Techniques

A kernel rootkit, DDefy.sys which was implemented by (Bilby, 2006), a disk filter driver to hide a file from the NTFS filesystem by faking its Directory Entry (Index), Master File Table (MFT) Entry MFT Data, and data in clusters for larger files. This is done by determining the drive info and NTFS characteristics for a partition, then locating filename, directory entry position on disk, clusters containing file data and their position on disk, then the data is hidden. Ddefy also intercepts the memory by performing a SSDT hook on NtMapViewofSection for all accesses to physical memory to modify the process and thread list. When a forensic tool wants to read the physical memory it does so by first acquiring the view of the memory in the virtual memory. Ddefy creates its own copy of file view thus making it appear as though the evidence has been gathered properly. This kind of modification would defeat memory analysis. A python script to demonstrate kernel mode AF techniques was written by (Stüttgen and Cohen, 2013) in which kernel debugger block hiding, hooking memory enumeration APIs (Application Peripheral Interfaces), and hooking memory mapping APIs were implemented. They evaluated these techniques against various memory forensic tools, and found that all of the tools were vulnerable to the aforementioned AF attacks which are discussed below.

An anti-forensic technique known as One Byte Modification also known as kernel debugger block hiding was implemented by (Haruyama and Suzuki, 2012). They scanned kernel objects by traversing through linked lists, for example, in the volatility framework this is done by searching for debug data header64 structure to find PsActiveProcessHead in kdd debug data64, and its address will contain the string “KDBG”.The authors then loaded a kernel driver into x86 XP virtual machine to modify that string value to demonstrate that memory analysis is vulnerable to that attack. A few of the forensic tools are dependant on KDBG to resolve symbols, and the aforementioned vulnerability effects the memory acquisition process can be demonstrated (Stüttgen and Cohen, 2013). It is not clear as to how

the kernel debugger block is hidden to disrupt the acquisition process as this attack does not hide any objects, rather it modifies the data in the kernel structures which defeats memory analyses. This has been demonstrated by installing the malicious driver and then acquiring the image successfully without any disruption to the memory acquisition process (Haruyama and Suzuki, 2012).

In hooking memory enumeration API the function MmGetPhysicalMemoryRanges() prints out the non-contiguous memory as the kernel sees it by omitting the device memory ranges. Memory acquisition tools use this function to determine whether it is safe to read certain memory locations or if it not safe to read in which case it will be a DMA (Direct Memory Access) mapped memory region. If this function is patched to return a null value, which indicates that this function has failed, the acquisition fails. Also, an attacker can hook this function to return certain memory ranges and hide certain memory ranges thereby defeating the memory acquisition process because to an investigator it will appear as though they have gathered the complete evidence without being aware that their acquisition process was incomplete and vulnerable by hooking MmGetPhysicalMemoryRanges() API. Whereas, in hooking memory mapping, the APIs ZwMapViewOfSection(), MmMapIOSpace(), and MmMapMemoryDumpMdl() are used to map the physical address space to the kernel's virtual memory in order to access physical address space. If the first two APIs are patched it might cause system instability because it is often used by memory driver tools, but a rootkit can hook these two APIs to thwart memory acquisition. The last API can be patched without cause system instability to prevent memory from being acquired.

(Milković, 2012) defeated windows memory forensics by developing a proof-of-concept (PoC) tool known as Dementia that hides objects in memory dumps. It was demonstrated that most of the memory forensic tools are vulnerable to attacks in the user-mode and kernel mode. The tool works well on Windows XP, Vista, and Windows7. It implements a user-model injection, and two kernel methods of hiding objects in memory dumps. If an object is to be

hidden using the kernel mode method, first the NtWriteFile() function is hooked using the filesystem minifilter. After the hook, the tool will check to see if a file written is a memory dump or not, and this is done by checking for known patterns that forensic tools exhibit such as such as specific NtWriteFile() arguments, process, drivers, file object values and flags. When the memory dump is detected, certain data can be hidden, for example, processes can be hidden by locating the EPROCESS block for those processes and unlinking them from various lists such as ActiveProcessLinks, SessionProcessLinks to name a few, and deleting the EPROCESS block itself. This method is so powerful that even volatility plugins such as psscan and psxview cannot detect the hidden process. The threads from target process are hidden by clearing the thread allocation, and deleting thread handles from PspCidTable. The author of Dementia claims that no application will detect those threads that have been deleted.

The process handles are hidden by unlinking the handles from the HandleTableList and deleted the "Obtb" allocation. The object header for a target process are traversed to see if PointerCount and HandleCount are equal to 1 which means that there any handles or objects created or opened by the target process, then the handle table entry and the object are deleted. If the objects are not opened by the target process then PointerCounter and HandleCount are decremented and the handle table entry is hidden because the target process is related to it even thought the object is not related to it. Also, the handle to the target process is stored in the handle table of PspCidTable and csrss.exe. The target handles are found and removed from the handle table to ensure that every artefact of the process is deleted in various memory locations. The next step is to hide the memory allocations which are described by Virtual Address Descriptor (DMA). VADs are stored in a self-balancing binary tree. The root of the tree is VadRoot and it is stored in EPROCESS. The tree is then traversed and the "VadX" descriptor is hidden. Then the entire memory region is cleared if the VAD describes the private memory or the process image. If the VAD describes a shared section of the memory

then this section is checked to see if it is used by the target process, and if it used then this region is cleared along with any mapped files such as file objects. Finally, the drivers are hidden by unlinking them from the PsLoadedModuleList, deleted the "MmLd" descriptor from the ldr data table entry, and clearing the driver image from the memory.

But there are other ways to detect drivers if they are hidden by this method. By using volatility one can scan for driver objects or symlinks to learn more information about the hidden drivers.

In the UserMode the attacker can hook WriteFile() and DeviceIOControl() APIs to modify the memory dump by hiding target process, process threads and connections. This is a bit harder to implement because one has no knowledge about the kernel addresses, there is no way to determine virtual address to private address translation, and only single pages of the dump is available. Therefore, all relevant information has to be determined manually from the dump. This is done by searching the current buffer for interesting allocations for the process, threads, and connections, and if target object encountered then the allocation is deleted. Also, if the object related to a target object (thread or connection) is found then it is deleted. The next difficult part is unlinking the process and the thread list because one cannot know where next or previous object is, only their virtual address (kernel) is known, and if that object was already written to the file there is no way anything could be modified in that buffer. To overcome this limitation the author, determined the virtual address of the object using self-referencing struct members in the EPROCESS structure (for example, ProfileListHead) then the object is cached in a dictionary with virtual address as the key, and noted the physical offset of that buffer in the dump, and then fixed the next or previous pointers either in the current buffer, or moved the file pointer to write a new value and restore the file pointer. The author states that win32dd is not vulnerable to user-mode attacks.

The assumption made by (Stüttgen and Cohen, 2013) that memory acquisition process can be trusted if the forensic tool does not rely on the operating system, is not completely true as Hidden in I/O Space (HIveS) which is an anti-forensic mechanism that does not depend

on the operating system to function, and it is capable of evading a large number of software based forensic tools (Zhang et al., 2018). HIveS is implemented on x86 AMD architecture. Physical address space is the is the range of memory addresses that is accessible by the x86 processors. The memory layout is configured by the BIOS by setting values in DRAM Base-Limit register pair. This configurations sets the physical address space which is mapped to DRAM in the north bridge. Any access (read or write) in this address space will be forwarded to the DRAM controller. The other set of registers that sets the memory layout are Mode Specific Registers (MSR) known as Top of Memory (TOM) registers and these registers are unique to the AMD processor architecture. There are two TOM registers TOP_MEM1 (TOM1) and TOP_MEM2 (TOM2) that set memory ranges in the address space. Any access within the range of ROM registers will be directed to DRAM controller, if the requests are out of TOM ranges then it is forwarded to the I/O space. The physical address layout of AMD architecture.

The purpose of TOM registers are that they help the operating system ascertain which memory space are backed by DRAM and I/O devices. Unlike the DRAM base-limit register which cannot be changed when the system is operational, the TOM registers can be changed to adjust the address layout, but it would not be wise to do so because the address layout was pre-determined by the BIOS during system initialisation, so the operating system will not be expecting the layout to be changed whilst the system is running, if one tries to change the layout using TOM registers it will result in a system instability and eventually the kernel will crash. However, the AMD architecture has another set of registers that is unique to its architecture. The Input Output Remap Registers (IORR). These registers have a specific functionality by which one can tweak the ranges of the physical address layout (even after the DRAM base-limit register was set during BIOS initialisation) whilst the system is operational without crashing the kernel. It is this property of IORR that HIveS leverages to accomplish its tasks.

When a processor core wants to access the memory it will first initiate a request through the north bridge. The multiplexer (MUX) in the north bridge decides whether to forward this request to DRAM controller, in which case, the process can then access the memory, or it forwards the request to the south bridge meaning that the processor core cannot have access to the memory. The MUX makes those decisions based on the physical address layout which was set by the BIOS base-limit register during system initialisation and then by the operating system by using TOM registers and IORR. The two states in which HIveS operates are locked and unlocked. When it operates in locked state, the processor, in this case core1 which is being used by the attacker, cannot access the memory since the requests will be forwarded to the south bridge. In the unlocked state, the malicious core can access the HiVes memory as the requests will be forwarded to the DRAM controller. The IORR registers consists of a base register and a mask register. The length of the memory region is stored in the mask register along with a valid bit which indicates the IORR configuration pair is active. Whereas the base register stores the starting address of the IORR region, and the two important flag bits, WrMem and RdMem. When WrMem and RdMem are set to 1, the north bridge forwards read or write requests for this physical address range to system memory. When these bits are set to 0, all requests are directed to the south bridge. In the unlocked state there could be a chance that the non-malicious core could be used by the forensic tool to access HIveS memory, to counter this issue the authors implement two new techniques known as Blackbox write and Translation Lookaside Buffer (TLB) camouflage, so that only the malicious core accesses the HIveS memory. The HIveS memory cannot be detected by forensic tools because HIveS operates outside the scope of the operating system, that is, it does not rely on the operating system to make changes in the memory. Hence making it harder for software forensic tools to detect its presence in the memory.

From sections 2.2 and 2.3 it is evident that various phases in the digital forensic process are vulnerable to anti-forensic attacks. This further highlights the problem from the findings of

2.1, that various phases of the digital forensic process appears to be affected by anti-forensic attacks, and most of the digital forensic models do not appear to factor in the anti-forensic techniques that affects the aforementioned digital forensic phases.
As this research work was motivated by the anti-forensic tool DECAF that was mentioned in 1.3, the next section demonstrates a hypothetical method on how it can affect the acquisition phase of the live forensic process.

2.4 Signature Based Detection

This section demonstrates a proof-of-concept that DECAF adopts to detect a memory forensic acquistion tool. For the purpose of this research APCO Guidelines (Williams, 2012) will be used as a guide to test if the live forensic process is vulnerable to AF attacks or not. The APCO guidelines recommends to use an USB device for the purpose of incident response/live forensics. In this research, it is argued that the usage of USB devices is not a secure option to gather evidence from a machine whilst it is switched on. The reason for this is that Windows is an event-based operating system (Russinovich et al., 2012a). Theoretically, whenever an USB device is plugged-in, various messages about the state of the USB device, such as device insertion and removal, is broadcasted to a window (Jacobs and Satran, 2017a). This message can be intercepted by AF tools and can perform various functions as discussed in 1.3, and can be practically demonstrated by conducting the following experiment. This demonstration intents to show the weakness in the live forensic process specifically during the acquisition phase:
Experiment 1: To detect a USB device when it is inserted to a live machine, and then perform the task of detecting any DF tool that is being run from the USB device, and when the both conditions are met, then programmatically shutdown/reboot the machine.
Software Tools: Windows 10 64-bit Virtual Machine using VMware Player 12.
Code-Lite version 10.0.0

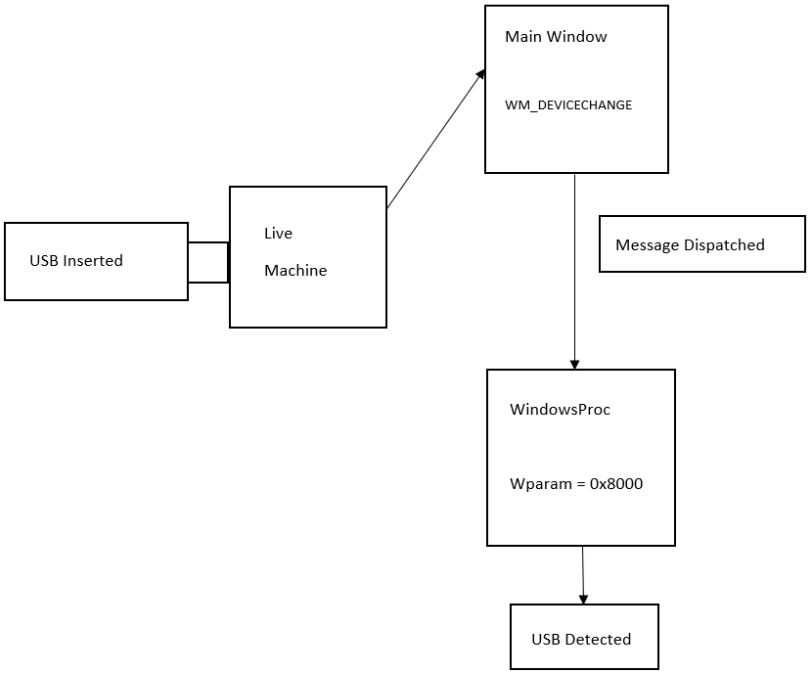


Fig. 2.1 Signature Based Detection of USB device

Method: Firstly, a C language-code was employed (Jacobs and Satran, 2017a) to detect a USB device the moment it is inserted into the machine. By capturing the broadcast message, in this case WM_DEVICECHANGE, with the help of a main window, and passing that captured message to the Windows Procedure Function (WNDPROC), so that a given operation can be performed. This would first detect the insertion of an USB device as shown in figure 2.1.

Step 2: Detecting the Forensic tool Methodology: After the USB device is inserted into the machine, the investigator will manually open the forensic tool or run an automated script to gather evidence in a forensic sound manner. When the forensic tool is run, it loads itself in the physical memory, and this circumstance can be detected.

First, a process list of the currently executing process are created by taking a snapshot of the system with the help of CreateToolhelp32Snapshot function. Then, the process list is traversed using the Process32First and Process32Next to determine if the target process, in this case a forensic tool, is in the process list. This is demonstrated by implementing a program Test.exe. It has 4 arguments h, d, p and s as shown in figure 2.2. “Test.exe h” is the

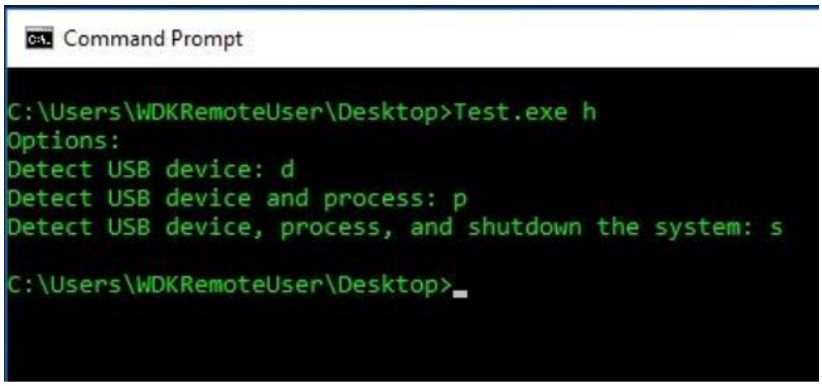


Fig. 2.2 Program to Detect USB Device

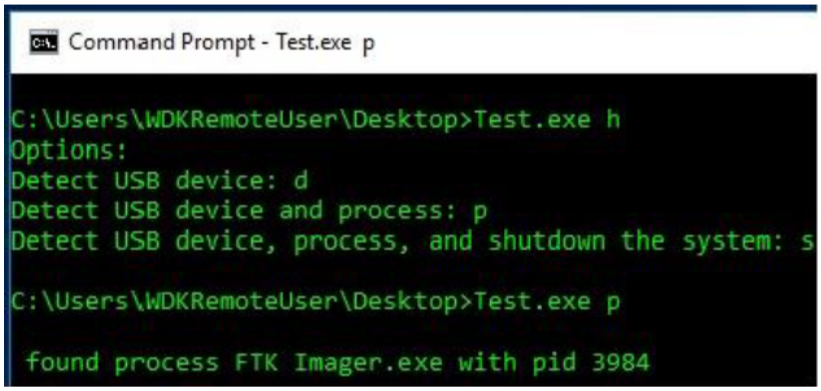


Fig. 2.3 Forensic Tool Detected

help option and shows the user about that usage of other options. To detect the whether a process is running when an USB is inserted just type in the “Test.exe p” command in the command prompt. Then, when an investigator inserts the USB device, first the USB device insertion will be detected, then the program checks whether a forensic tool is running or not. If the tool is not running then it will display “process does not exist” on the screen, or if the process exists it displays the name of the process along with its process identifier (pid), as shown in figure 2.3.

Step 3: Shutting Down the machine

After the insertion of the USB device is detected, and the forensic tool detected, the machine can be shut down. Now, why would one want to shut down the system? briefly mentions and describes the shortcomings of the traditional forensic process of pulling the plug off the

system. To overcome those shortcomings such as anti-forensic tool executes only in memory, the system cannot be taken down due to business critical operations which could otherwise lead to legal ramifications if it is shut down. Also, if the investigator decides to image the hard drive, the integrity of the evidence would be compromised, and later this evidence could be deemed to be inadmissible in the court of law on the grounds of spoliation of evidence, because during shutdown the operating system flushes data onto the hard drive, or in worst cases if the attacker has written a script to wipe the hard drive during shut down then there would be no evidence to work with. Thereby not only defeating live investigative process, but also the traditional investigative process. To programmatically shutdown the system, the ExitWindowsEx function is used. The program must enable the SE_SHUTDOWN_NAME privilege must be first enabled, and this is done through AdjustTokenPrivileges function (Jacobs and Satran, 2017b).

The shortcoming of signature based detection is that, if the process name of the forensic tool is changed, then the method described in this section to detect a forensic tool will fail. Therefore for that reason, this study intended to explore alternative ways to detect forensic memory acquisition such as pattern matching and detection, which gave rise to the following research question (RQ1):

How can forensic memory acquisition be accurately detected by artificial intelligence techniques? (or)

Can machine learning algorithms and deep learning models be able to recognise or detect forensic memory acquisition patterns?

The objective to the corresponding research question is to find the application of AI to the domain of digital forensics and anti-forensics. In the next section the application of AI to those domains are reviewed.

2.5 Artificial Intelligence in Digital Forensics

Artifical intelligence (AI) is complex to define as it is comprised of numerous definitions in the literature (Dönmez, 2013, Pg 9). Broadly, the definition of AI can be divided into two categories. The first category involves observing and hypothising human behaviour, and the second catergory involves a rationalist approach which is a mixture of mathematics and engineering (Russell and Norvig, 2010). For the relevance in digital forensics, (Duce et al., 2010) adopt a pragmatic approach and define AI as, "creating a computer process that acts in a manner that an ordinary person would deem intelligent". It appears that various researchers have applied this definition of AI into their framework to design AI applications for digital forensic purposes. These AI applications mostly utilise concepts from machine learning (ML) (Stephen, 2014, Pg 6) and deep learning (DL) (Dönmez, 2013, 388), one of the few branches of AI, to detect and classify malwares, improve intrusion detection systems, combat cyber crimes to name a few. These various applications of AI will be reviewed in the following section.

2.5.1 Using AI to Detect Malware

One of the areas where ML is used in digital forensics is malware detection. During the past few years researches have used various ML algorithms to detect malware. One of the ways by which malware can be detected is by extracting features from the header of portable executable 32-bit file on Windows operating system. The experimental results of (Markel and Bilzor, 2015) show that decision tree classifier perform better by discriminating between malicious and benign executables over logistic regression and naive bayes algorithms. A similar approach of extracting features from malicious and benign excutables was carried out by (Sewak et al., 2018). In their experimental setup features were gathered from executables by autoencoders and by varying threshold. Later, the features were inputted to random forest classifier (RFC) and a deep neural network (DNN) with two, four, and seven layers

respectively. The results show that RFC performs better than any of the DNNs with 99.78 percent accuracy using varying threshold.

Another well known way to detect malware is through memory images. Features can then be extracted from memory images by using wavelet transforms. Makandar and Patrot (2017) had adopted such method to extract features. Afterwards, principle component analysis was applied to the extracted features to select the desired features. Their results show an accuracy of 98.84 percent using k-neighbours algorithm and 98.88 percent using support vector machine (SVM) algorithm to classify malwares. Whereas, (Dai et al., 2018) derived their memory dumps from sandbox whilst a malware is running dynamically on the virtual machine. Later the memory dumps were converted to grayscale images for feature extraction. Histogram oriented gradients (HOG) were used to extract features from grayscale images. Their results show their multi-layer perceptron (MLP) model classifies malware with 95.2 percent accuracy, and with a f1-score of 94.1 percent.

Instead of extracting features from executables or converting memory dumps to images, (Sihwail et al., 2019) directly extract two types of features from memory dumps. The first feature is extracted with volatality analyser through which API features are extracted, and from there memory features vector are created. The second feature is extracted using cuckoo sandbox and by monitoring the log files, the dynamic features vector are created. These two features are first classified individually, and then they are integrated to be classified and compared with five ML algorithms. When classified individually, the results show that dynamic features perform well over memory features with SVM being the leading ML algorithm with accuracy of 97.4 percent and false positive rate of 4.9 percent. Now, when both the features are integrated the overall performance of the classification is increased by using SVM with accuracy being 98.5 percent and FPR of 1.7 percent. Whilst, the aforementioned malware detection methods mostly use ML algorithms, (Le et al., 2018) convert their malware dataset to grayscale images and use convolutional neural network (CNN) and its their two

improvised version of CNNs, namely CNN Uni Long Short Term Memory (CNN UniLSTM) and CNN BiLSTM which consists of one and two layers of LSTM respectively just before the output layer. Their results show that CNN BiLSTM outperforms other models with an accuracy of 98.2 percent. Similarly, (Vinayakumar et al., 2019) also adopt various DNN architectures to compare and contrast their results with ML agorithms. After the dataset was converted to grayscale images, the results show that two layered (CNN2-LSTM) performed better with 96.3 percent accuracy for dataset 1, and 98.8 percent accuracy for dataset 2 using 10-fold cross validation.

One of the innovative ways to detect malware is to extract feature from malware network traffic instead of malware executables or binaries. The method of extracting features from malware traffic was adopted by (Chen et al., 2019). Various features were extracted from malware traffic, and feature selection techniques such as analysis of variance (ANOVA) and AutoEncoder were used. To account for data imbalance, the synthetic minority over-sampling technique (SMOTE) was used. Finally, extreme gradient boost (XGBoost) ML algorithm was to used to classify malware. The experiments were carried out using XGBoost with ANOVA, AutoEncoder, ANOVA and SMOTE, and finally AutoEncoder and SMOTE which has the highest classification accuracy of 99.88 percent when compared to other methods.

Whilst answering RQ2, it was found that AI is being applied to the digital forensics and cyber security domain as seen in 2.5.1. It does not appear that AI is being applied to the anti-forensics domain, and this is the knowledge gap this study has identified. To fill this knowledge gap, RQ2 was answered by adopting the methodology in 3.3, and supporting the hypothesis in 3.3.2 with the results obtained in chapters 4 and 5 respectively.

2.6 Overview of Machine Learning Algorithms

In this section, the eight supervised machine learning algorithms that have been used in this research study are briefly discussed. These eight algorithms have been selected as they

are extensively used in the literature of cyber forensics and security as seen in the previous section 2.5.1.

2.6.1 Linear Discriminant Analysis

The linear discriminant analysis (LDA) is model that estimates mean and variance from a dataset for every class. For example, if only one variable is considered with two classes, then the mean (m), for each input variable (x), for each class (c) is estimated by using equation 2.1.

$$m_c = \frac{1}{n_c} \times \Sigma_{i=1}^{n}(x_i) \tag{2.1}$$

Where m_c is the mean of the dataset (x) for the class (c). n_c is number of instances with class (c). The variance is estimated as the average squared difference of (x) from the mean, as shown in equation 2.2

$$\sigma^2 = \frac{1}{n-c} \times \Sigma_{i=1}^{n}(x_i - m_c)^2 \tag{2.2}$$

Where, σ^2 is the variance across the input feature (x), n is the number of instances, c is the number of classes, and m_c is the mean of the input features (x) for the class to which x_i belongs to. This means the squared difference of each input value from the mean within class groups is calculated but with the average of the differences across all class groups. LDA predicts outputs by estimating the probablity that a set of input features belongs to each class, then the class with the highest probability is outputted. This model uses bayes theorem to estimate probabilities. The formula used to estimate probability of the output class (c) given the input feature (x) is expressed in equation 2.3.

$$P(Y = c|X = x) = \frac{P(c) \times P(x|c)}{\Sigma_{i=1}^{C}(P(i) \times P(x|i))} \tag{2.3}$$

Where, P(Y=c|X=x) is the probability of the class Y=c, given the input features x.
P(c) is the probability of a class (c).

P(x|c) is the estimated probability of x belonging to the class (c).

The term in the denominator is accounted for each class (i). It is the probability of the i^{th} class P(i) and the probability of the input feature given the class (i), that is, P(x|i).

To estimate P(x|i), gaussian distribution function can be used, and during estimation the probability terms cancel each other, and it results in a discriminate function for class (c) which is calculated for each class. The class with the greatest discriminant will make the output classification (Y=c). The equation of the discriminant function is:

$$D_c(x) = x \times \frac{m_c}{\sigma^2} - \frac{m_c^2}{2 \times \sigma^2} + ln(P(k)) \tag{2.4}$$

$D_c(x)$ is the discriminant function for class (c) given input feature x, the mean (m_k), variance (σ^2) and P(c). All these terms are estimated from the input features. The ln() function is the natural logarithm (Jason, 2016, Pg 63).

2.6.2 Support Vector Machines

Support Vector Machines (SVM) is one of the popular and well known machine learning developed in the 1990s. The working of SVM is best explained by the concept of maximal-margin classfier. Consider, the input features (x), that is, the data which is the coloumns in the dataset for an n-dimensional space. For example, if the dataset contains two input features, then this would form a two-dimensional (2D) space. Now, in order for the classification to take place, the points in the space must be separate. In the case of 2D space, a line which is also known as hyperplane, can be used to separate the points in the input feature space by their class. In 2D space this will be class 0 and class 1. The equation of the hyperplane in 2D space is represented by equation 2.5

$$B0 + (B1 \times X1) + (B2 \times X2) = 0 \tag{2.5}$$

The coefficients B1 abd B2 determine the slope of the line, and the intercept B0 is found by the learning algorithm. X1 and X2 are two input features. Classification can be made by substituting the input feature values in equation 2.5, and then calculate whether a point is above or below the line. Whether a point in space belongs to class 1 or class 0 is determined by the following steps:

1. If the hyperplane equation returns a value greater than zero, then the point lies above the plane and it belongs to the first class, which is class 0.

2. If the hyperplane equation returns a value less than zero, then the point lies below the plane and it belongs to the second class, which is class 1.

3. If the hyperplane equation results in a value which is almost zero, then it may not be possible to classify that point.

4. If the result in equation 2.5 is large, then the model's prediction may have a higher confidence.

The distance between the line and closest data point is known as margin. A Maximal-Margin hyperplane is a line with largest margin that can differentiation between two classes. This margin is calculated as the perpenducular from the line to the closest point. These points are relevant in defining the line and the classifier, and are known as support vectors. The hyperplane is learned from the training dataset by maximizing the margin (Shalev-Shwartz and Ben-David, 2013).

In real-time scenarios, the data cannot be separated by the hyperplane perfectly. This is due to restriction placed on the hyperplane to maximize the margin. In order to minimize the restriction a method known as soft margin classifier (SMC) is adopted. SMC allows certain points in the training dataset to breach the separating line by introducing coefficients (also known as slack variables) to give the margin wiggle room in each dimension. Therefore, increasing the complexity of the model.

The magnitude of the wiggle is defined by the tuning parameter C across all dimensions. This means C defines the number of violations the margin is allowed. If C is 0, then there are no violations, and the classification is done based on the maximal-margin classifier. When the hyperplane is being learned fromm the training dataset, the training data points that lie withhin the distance of the margin affects the placements of the hyperplane, which means the value of C influences the number of support vectors used by the classifier. Larger C values will result in lower variance and higher bias because it makes the algorithm less sensitive to the training data. Whereas, smaller C values will result in higher variance and lower bias because the algorithm becomes highly sensitive to the training data (Xin et al., 2018).

2.6.3 Gaussian Naive Bayes

In Gaussian Naive Bayes (NB), in addition to the probabilities, the mean and standard deviation of the input features (x) are first calculated for each class. The mean for each input feature is:

$$m(x) = \frac{1}{n} \times \sum_{i=1}^{n} (x_i) \tag{2.6}$$

Where x are the values for an input featurein the training dataset and n is the number of instances. Standard deviation is calculated by the following equation:

$$SD = \sqrt{\frac{1}{n} \times \sum_{i=1}^{n} (x_i - m(x))^2} \tag{2.7}$$

Where, n is the instances, x_i is the value of the input feature (x) for the i^{th} instance and m(x) is the mean. Predictions are made based on the Gaussian Probability density function (pdf). Gaussian pdf gives the probability estimate for a input feature of a class. The equation of Gaussian pdf is

$$pdf(x, m(x), SD) = \frac{1}{\sqrt{2 \times \pi} \times (SD)} \times e^{-(\frac{(x-m(x))^2}{2 \times SD^2})} \tag{2.8}$$

Where, x is the input data, mean, and SD are explained in equations 2.6 and 2.7 respectively, π is the numerical constant whose value is 3.14, e is the Euler's number. Probabilities can then be substituted in equation 2.8 to make predictions (Jason, 2016, Pg 94).

2.6.4 K-Nearest Neighbours

K-Nearest Neighbours (KNN) makes predictions by using the entire dataset. When a new dataset is received as an input to the algorithm, predictions are made by searching through the entire training dataset for the K most similar events, that is, the neighbours, and summarising the output class for those K events. During classification this is the mode (the most common feature occuring in the dataset) class value. In order to determine the K instances in the training set that are closely similar to a new input feature in the test dataset, a distance measure is used. The most popularly used distance measure is Euclidean distance. It is calculated as square root of thesum of differences squared between two points x_1 and x_2 across all input features i. The formula for Euclidean distance (ξ) is

$$\xi(x_1, x_2) = \sqrt{\sum_{i=1}^{n} (x_1 - x_2)^2} \tag{2.9}$$

Other distance measures are the Hamming, Manhattan, and Minkowski Distance. In hamming distance the distance between the binary vectors are calculated. Manhattan distance calculates the distance between vectors using their absolute difference. The genralisation of Manhattan and Euclidean distance is Minkowski distance (Xin et al., 2018).

2.6.5 Decision Tree

Decision tree uses one of the most powerful datastructure known as binary tree. Binary trees not only have low computational cost but also the cost of using it is even lower, that is, the time taken to implement a decision tree algorithm is lesser. For this reason it is used

in machine learning as querying a trained algorithm should be quick. A benefit for using decision tree aclgorithms is that there is transparency to get to the classification answer by following a tree. This, in turn increases the trust in the algorithm, and is the reason why it has become popular over recent years (Stephen, 2014, Pg 249). Few of the well known decision tree algorithms are Iterative Dichotomiser 3 (ID3), C4.5, and Classification and Regression Tree (CART).

In ID3, the data set (S) is trained to produce a decision tree. This process entails two steps which are entropy and information gain. Entropy H(S) is a measure of uncertainity in the data set (S), and is given by the equation:

$$H(S) = \sum_{x \in C} -p(i) log_2 p(i) \tag{2.10}$$

Where,

- S is the dataset for which entropy will be calculated. The dataset changes at each step of the algorithm, that is, at each node.
- C is the set of classes in S.
- p(i) is the proportion of the number of items in class i to thee number of items in S.

When, H(S) is zero, the dataset (S) is perfectly classified, that is, all items in S are of the same class. The entropy is calculated for all attributes. The attribute with smallest entropy is used to split dataset (S) on an iteration. After entropy, the information gain is calculated. The information gain $G(A)$ of an attribute "A" is the difference in entropy in dataset (S) and the dataset (S) given the attribute A.

$$G(S,A) = H(S) - \sum_{f \in F} p(f) H(f) = H(S) - H(S|A) \tag{2.11}$$

In equation 2.11,

- H(S) is the entropy in dataset (S)
- F is the subset created after splitting S by attribute A
- p(f) is the proportion of the items in f to the number of items in S
- H(f) is the entropy of subset f
- H(S|A) is the entropy of S given A

In ID3 information gain is calculated for each attribute. The attribute with highest information gain is used to split *S* on the current iteration. In CART, the Gini index or Gini Impurity is used as the information measure. Here, the impurity means that the decision tree has each leaf node represent a set of instances that are in the same class in order to avoid mismatches (Stephen, 2014, Pg 260). For any attribute a, the Gini impurity is calculated as:

$$G_a = \sum_{i=1}^{c} \sum_{j \neq i} N(i)N(j) \tag{2.12}$$

Where c is the number of classes. i and j are specific classes. Since there has to be an output class, $\sum_{j \neq i} N(j) = 1 - N(i)$, then equation 2.12 can be written as:

$$G_a = 1 - \sum_{i=1}^{c} N(i)^2 \tag{2.13}$$

The information measure can be calculated in terms of misclassification rate (Stephen, 2014, pg 261). This can be done by adding a weight to the misclassification instances. The concept here is to evaluate the cost of misclassifying a datapoint in class i and class j, and then add a weight (λ_{ij}) that reflects the misclassification. Now, equation 2.12 can be written in terms of misclassification rate as

$$G_i = \sum_{j \neq i} \lambda_{ij} N(i)N(j) \tag{2.14}$$

2.6.6 Random Forest Classifier

Random Forest Classifier (RFC) is an extension to the decision tree classifier. The shortcoming of decision tree classifier is that it suffers with high variance as it overfits the data. In order to overcome this shortcoming RFC is adopted. RFC makes use of two concepts to minimize overfitting, which are random sampling of training datapoints during tree-building, and random subset of features when splitting nodes.

During training, a tree in a random forest learn from the random sample of the dataset. The samples are drawn with replacement and is known as bootstrapping, and it means some samples will be used will be used on a tree multiple times. By training trees on different samples, and if the trees have high variance when compared to a particular set of training dataset. Overall, the entire forest will have lower variance but not at the cost of increasing the bias.

The other concept in RFC is that only a subset of features will be considered in each decision tree for splitting a node. In sklearn RFC library, which is used in this research, this is controlled by adjusting the parameter sqrt(n_features) for classification. For example, if there are 9 features, then only 3 features would be selected to split a node. Now, during the testing phase the predictions from each decision tree is averaged to make the final prediction. The process of training each individual decision learning algorithm on a subset of data is known as bootstrap aggregating or bagging (Jason, 2016, Pg 126).

2.6.7 AdaBoost Classifier

AdaBoost is mostly used to boost the performance of decision trees. It is also referred to discrete AdaBoost when applied to classification problems. It works good with weak learners, which are models that achieve accuracy greater than random chance for a classification problem. The reason one-level decision trees, also known as decision stump, are most commonly used with AdaBoost is because they contain only one decision to make a classification. As

each instance in the training dataset is assigned a weight, and the initial weight is set to:

$$w_i = \frac{1}{n} \tag{2.15}$$

Where i is the i^{th} training instance, and n is the number of training instances.
Using the weighted samples a weak classifier, that is, a decision stump is formed from the training dataset. For example, in binary classification situations, a decision stump takes an input feature and makes one decision, and then outputs $+1.0$ or -1.0 for class 1 (first class) and class 0 (second class) respectively. For the trained model, the misclassification rate or error (ε) is calculated as:

$$\varepsilon = \frac{t_i - N}{N} \tag{2.16}$$

Where, t_i are tje training instances that are predicted correctly by the model, N is the total training instances. To calculate the overall error, equation 2.16 can be modified to use the weights of the training instances as shown in equation 2.17:

$$\varepsilon = \frac{\sum_{i=1}^{n}(w_i \times p\varepsilon_i)}{\sum_{i=1}^{n}}(w_i) \tag{2.17}$$

Equation 2.17 is the weighted sum of the misclassification rate. Where, w_i is the training instance weight, $p\varepsilon_i$ is the prediction error for i^{th} training instance. In the case of misclassification $p\varepsilon_i$ is 1 and 0 in the case of correct classification. The next step is to calculate the stage value which provides weights for the predictions AdaBoost makes. The stage value (α) is calculated as follows:

$$\alpha = ln(\frac{1-\varepsilon}{\varepsilon}) \tag{2.18}$$

Where, α is the stage value used for weighting predictions from the model, $ln()$ is the natural logarithm, and ε is the misclassification error for the model. Stage weight is more for accurate models. The training weights are updated during every iteration and this gives more weight

to incorrectly predicted instances and less weights to instances that have been incorrectly predicted. The updated training instance weight (w) is calculated as:

$$w = w \times e^{\alpha \times p\varepsilon_i} \tag{2.19}$$

The final step is to make predictions by calculating the weighted average of the weak classifiers. Given a new input, the predicted values are weighted by each weak learners stage value. The prediction for the ensemble model is taken as the sum of the weighted predictions. If the sum is positive, then class 0 is predicted, and if it is negative then the class1 is predicted. The prediction is calculated as

$$P(x) = sgn(\alpha \sum_{i=1}^{N} p_t(x)) \tag{2.20}$$

Where, $P(x)$ is the output classifier or prediction, α is the stage value, N is the number of training instances, and $p_t(x)$ is the prediction from the decision stump (Jason, 2016, Pg 137).

2.6.8 Gradient Boosting Classifier

Gradient boosting combines the weak learners sequentially to a strong learner by an iterative process. The goal here is to teach a model F, to make predictions of the form $\hat{y} = F(x)$ by minimizing the means squared error (MSE) $\frac{1}{n}\sum_i(\hat{y}_i - y_i)^2$, where i iterates over the training dataset consisting of n samples and the output predictor variable y.

At each iteration m of gradient boosting, it is assumed that the imperfect model F_m, which initially is a weak model that predicts the mean of the output variable y. On every iteration the gradient boosting algorithm improves on F_m by sequentially constructing new models that adds the estimator h to improve the model after each iteration. This improvement is

mathematically expressed as:

$$F_{m+1}(x) = F_m(x) + h(x) = y \tag{2.21}$$

As gradient boosting will try to fit h to $y - F_m(x)$, F_{m+1} attempts to rectify errors of its previous model F_m. Therefore, the final model F_m will result in higher accuracy with minimum amount of error (Chen and Guestrin, 2016).

2.7 Conclusion

In section 2.1, various DFMs were reviewed to find out whether anti-forensic affects are taken into account in their approach or design. The findings show that there are generic and specific DFMs proposed, and it has been identified that AF affects have not been taken into consideration in DFMs apart from (Rekhis and Boudriga, 2012a) and (Rani and Kumari, 2017). This further necessitated to review various AF techniques 2.3, to find which AF techniques affected which phase in the DF process. The aforementioned, in conjunction with the findings from section 2.2, in which forensic acquisition tools were compared to see if they were resistant or vulnerable to AF techniques. The findings identified the research problem, that the four major phases of the DF process acquisition, examination, analysis, and reporting are affected by AF techniques. To address this research problem, RQ2 was posed: How can digital forensic models be validated when they are affected by anti-forensic techniques? and was answered with help of the research objective 6 (RO 6) mentioned in section 1.4 in chapter 6.

The findings from section 2.5.1 show that one of the most prominent application of AI in digital forensics is detecting malware, and various methods have been proposed to detect malware. The most common method being, first transforming a memory image into the corresponding gray-scale image. After this transformation, majorily ML algorithms were

applied to classify various malware families. Mostly, the authors presented their results based on the accuracy of the results, and overlooked the fact accuracy alone is not enough to analyse a ML model, and various other factors such as precision, recall to name a few must be taken in account.

AI is mostly applied to assist in digital forensic examinations such as detection of malware in a memory image as seen in section 2.5. Its application in anti-forensics is scarce. Also, the anti-forensic techniques in section 2.3 is complex and works only on Windows 32-bit machines in user-mode, and they would cause a BSOD when implemented on Windows 10 64-bit machines. When coupled with the shortcoming demonstrated in section 2.4, it highlights the knowledge gap this research has identified. Which is, the application of AI to the domain of anti-forensics. Specifically, the application of machine learning algorithms to detect memory acquisition patters during a live forensic acquisition process. To fill, the aforementioned knowledge gap, the research question (RQ1) was posed: How can forensic memory acquisition be accurately detected by artificial intelligence techniques? RQ1 was answered with the help of research objectives (RO1-RO5) mentioned in section 1.4 and by adopting and justifying the research method discussed in Chapter 3, and proposes two methods as discussed in chapter 4 and 5 respectively, in which various supervised machine learning algorithms were used to detect the forensic memory acquisition patterns.

Chapter 3

Research Methodology

3.1 Introduction

This chapter outlines the research approach that is taken in AI research in section 3.2 and compares it with the scientific method. In this research, the research approach gave rise to the methodology and that in turn had influenced the choice of research method, which is elaborated in section 3.3. The research problem and the central research question are stated again to operationalize the hypothesis, and this further guided in designing the experiment. Data collection and pre-processing is discussed in detail as it is an important step as they form the inputs to ML classifiers. Various evaluation metrics are presented and explained as the ML and DL models will be evaluated based on those metrics.

3.2 Research Approach in Artificial Intelligence

In AI, the dominating epistemological stance is the positivist approach (Alauthman, 2016, Pg 6), where the focus of the study is on facts to determine the causal relationship between independent variable(s) and dependent variable(s) (Gray, 2014). In AI, the research method-

ology that emerges from the positivist approach is the experimental research method, and it consists of the following steps as mentioned by (Cherkassy, Pg 5):

1. State the problem: The problem in AI is application specific which is performed in a particular domain of AI such as machine learning and deep learning. The domain specific knowledge and experience is the driving force in figuring out a problem in AI. Focus here must be on finding and framing an effect problem statement rather than focusing on the learning algorithm.

2. Formulate the hypothesis: Once the problem has been identified, the next step is to formulate a hypothesis that can be tested and falsified. The hypothesis should be formulated in such a way so that the independent and dependent variables are identified.

3. Design the experiment: In this step the data is generated by selecting an experimental design procedure. In AI experiments, this could be a true experimental procdure or quasi-experimental procedure. A true experimental procedure is an experiment where data is generted randomly, whereas in a quasi-experiment the data assignment and generation is not random. Proper selection of experiment will influence the sampling size which is crucial for collecting data.

4. Collect and pre-process the data: Once the data is generated, it should be processed in a format that is desired by the AI algorithm.

5. Estimate the model: When the hypothesis in step 2 was formulated, the variables in it must be indentified, so that the dependency between the input and output variables or features can be determined. This is done by qusntifying the dependencies from available data and the previous knowledge about the problem. The aim here is to design models for accurate prediction of future outputs from the known input variables. Predictive accuracy is also known as generalization capability in neural networks.

6. Interpret the Models: Depending on the problem and hypothesis, the researcher will be informed as to what metrics must be evaluated so that the model can be interpreted. Here interpretation is done to determine whether the model has higher accuracy, precision, recall to name a few.

The steps mentioned in 3.2 by (Cherkassy, Pg 5) is similar to the scientific method, and consists of the following steps, (Goebel and Plagemann, 2009):

1. Frame questions: The questions are framed from the existing theory and observations. In a research study, the researcher might be informed by the literature review as to what questions needs to be framed to address the knowlege gap or the problem that has been identified.

2. Formulate Hypothesis: Formulating hypothesis is an important step in the scientific method. A hypothesis is a conjecture because is provides a provisional or a tentative explanation regarding a certain phenomenon. Again, in a research study the researcher is informed by the research questions in step 1 to formulate the hypothesis. An importaint point to note here is, if the hypothesis formulated in this step is a scientific hypothesis, then it must be falsifiable. It means that there are certain outcomes in an experiment where the evidence will not support the hypothesis. If a hypothesis cannot be falsified, then it will not satisfy the criteria of scientific hypothesis, (Jeong and Kwon, 2006).

3. Make Predictions: After formulating the hypothesis, predictions are made as a consequence of logical deductions. The purpose of prediction is to compare the results of the experiments to that of the predictions. Whilst making predictions the answer should not be known, or else it will weaken the evidence.

4. Test the hypothesis: The hypothesis is tested by conducting experiments. The purpose of the experiment is to determine whether the results of the experiment agree or

disagree with the predictions. Agreement increases the confidence or supports the hypothesis. It does not mean that the hypothesis is true. Experiments must be designed to minimize the effects of variables other than the independent variable, (Gray, 2014). Steps 2-3-4 must be looped until the agreement between the results and predictions are reached. If major discrepancies are found, the researcher must go back to step 1.

5. The hypothesis becomes a theory when it has obtained consistency by withstanding rigourous attempts to falsify it. This will give rise to a set of propositions that define a new phenomena or a theoretical concept.

3.3 Proposed Research Methodology

As seen in section 3.2, the AI research method and scientific method are similar, and steps 2, 3, and 4 are iterative in the latter method. In this research, the research methodology is informed by the epistemology of positivism, and is influenced by the methods discussed in section 3.2. Therefore, the research methodology adopted in this research is the experimental research method which consists of the following steps:

1. Identify the research problem and pose question(s)
2. Formulate the hypothesis and make predictions
3. Design the Experiment
4. Gather and pre-process the data
5. Evaluate and Interpret the model
6. Accept, modify, or reject the hypothesis

The first step will be to identify the research problem and then to address the problem various research questions were posed from which a hypothesis was formulated as a tentative answer

that was tested by conducting an experiment. Just before conducting the experiment, the sample size was determined in a way that reduces type I and II erros. Then an experiment was designed to gather and pre-process the data. Various evaluation metrics are presented in section 3.3.6. The hypothesis was then modified, accepted and falsified based on the results presented in chapters 4 and 5. The sections that follow describe each step of the proposed research method in detail.

3.3.1 Problem statement and Research Questions

The research problem was identified in section 2.7 and to address that problem a research question was posed. To answer the central research question 1 (RQ1), "How can forensic memory acquisition be accurately detected by artificial intelligence techniques?", from the findings of the literature review, it was found that AI is being applied to the AF domain. Therefore to meet the objective (RO1-RO5), the research focuses on detecting the live forensic acquisition patterns on windows 10, 64-bit machines using supervised machine learning algorithms.

This research proposes the following method to capture memory acquisition patterns. Since a process is a software program that executes in the physical memory. Every process on a Windows machine has certain CPU parameters Rodola (2019) and, memory and I/O parameters associated with it as described in (Russinovich et al., 2012b, Pg187), and (Russinovich et al., 2012b, Pg 8) respectively. The forensic memory acquisition tools such as AccessData FTK Imager (FTK, 2019), Magnet RAM Capture (MAG, 2019), Belkasoft RAM Capturer (Bel, 2019) to name a few are also processes when they are executed on the machine to acquire memory. Since these memory forensic acquisition tools will have their own memory, I/O, and CPU (MIOC) parameters associated with them. Therefore, to address the problem mention in section 1.4 and answer RQ1, the following research hypothesis was

formulated in sub-section 3.3.2 as a tentative answer that was tested using the experimental research method described in sub-section 3.3.4.

3.3.2 Hypothesis

The hypothesis is stated as follows: It may be possible to detect forensic memory acquisition on a windows 64-bit machine by detecting the variability pattern of their respective MIOC parameters whilst memory is being acquired by a forensic tool.

In the hypothesis, it was assumed that this variability pattern in MIOC parameters is due to memory acquisition. Another assumption is that this variability pattern is distinct and native to memory acquisition and non-acquisition, and hence variability pattern of MIOC parameters is termed as as Distinctive Native Attribute (DNA) patterns, which is defined as: The distinct MIOC patterns native to a memory forensic acquisition tool. It is distinct because it is different for memory acquisition and non-acquisition and native in the sense that each forensic tool has its unique associated MIOC parameter variation when executed in memory.

3.3.3 Prediction

From the hypothesis formulated in section 3.3.2, the argument this study makes is that there is a causal relationship that exists between memory acquisition and MIOC parameters, because memory acquisition induces DNA patterns in MIOC parameters, and it is these DNA patterns in MIOC parameters that can differentiate between memory acquisition and non-acquisition. Therefore, if memory acquisition is the cause for variability in MIOC parameters, then by the concept of inverse function MASH (2005) MIOC parameters can be mapped back to memory acquisition and is explained in Prediction 3.1.

Prediction 3.1: If memory acquisition (M_a) is a function (f) of MIOC parameters, which is

mathematically expressed in equation 3.1 and diagramatically in figure 3.1 :

$$f(M_a) = MIOC \tag{3.1}$$

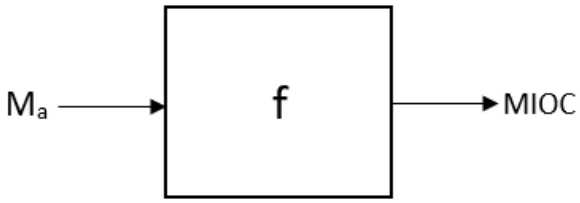


Fig. 3.1 MIOC Parameters as a function of Memory Acquisition

And, if there exists an inverse function such that,

$$f^{-1}(MIOC) = M_a \tag{3.2}$$

then memory acquisition is said to be detected, which in turn means memory acquisition causes DNA pattern effect in MIOC parameters. In other words, when MIOC parameters form the inputs to a ML classifier as shown in figure 3.2, and if the inverse function (f^{-1}) can differenciate between memory acquisition and non-acquisition by conforming to the threshold criteria in section 3.3.7, then memory acquistion can be detected.

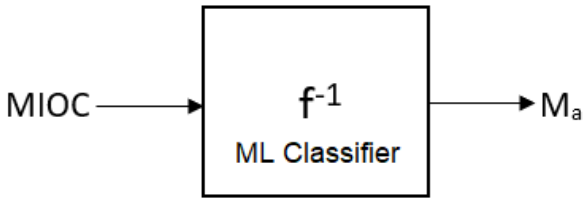


Fig. 3.2 Inverse Function of MIOC

But during detection, there exists an error (e) which threatens the accuracy of the outcomes (Jason, 2016). Lower the value of e, higher the confidence in the evidence that supports the hypothesis. Greater the value of e, then the hypothesis may be modified or rejected. The

error (e) is described in detail in section 3.3.6. Therefore, equation 3.2 can be re-written as,

$$f^{-1}(MIOC) = M_a + e \tag{3.3}$$

To further support the hypothesis, DNA patterns from MIOC parameters are extracted by the method described in section 5.2.1. These patterns are then classified into memory acquisition and non-acquisition using 3L-CNN described in section 5.4.

3.3.4 Experimental Design

From the hypothesis, it is clear that this research concerns with the causal relationship between memory acquisition (independent variable) and MIOC parameters (dependent variables). Therefore, to determine the existence of causal relationship between memory acquisition and MIOC parameters, two groups are made. The first group is the control group in which memory was not acquired, and the second group is the experimental group in which the memory was acquired. The MIOC parameters was gathered from ten windows 10 64-bit machines. Before collecting these parameters from the machines, it is essential to determine and justify the sample size, which is discussed in the following section.

Sample Size Justification

The minimum sample size is determined by performing power analysis. Power analysis comprises of four important paramters effect size, significance level, statistical power, and sample size (Becker, 2011, Pg 56). In statistical hypothesis testing, the null hypothesis (H_0) predicts the outcome of an experiment. For example, the null hypothesis for a student t-test is there is no significance difference between the means of two groups. This test is interpreted using the significance level (α) also known as the p-value. The p-value is the probability of observing the outcome given that the null hypothesis is true. Usually, the significance level is

set to 0.05 or 5 percent in most experiments. Given the significance level, two types of errors can occur whilst interpreting the results (Becker, 2011, Pg 50-52):

Type I error: It is also known as false positive(FP), and it occurs when the null hypothesis is rejected and there is no significant effect.

Type II error: It is also known as false negative (FN), and it occurs when the null hypothesis is not rejected and there is significant effect.

To determine if the probability of a test correctly rejects null hypothesis then statistical power test needs to be conducted. Statistical power is a probability that a test will correctly reject null hypothesis and it has relevance only when the null hypothesis is false, (Becker, 2011, Pg 60). If the statistical power for a given experiment is higher, then lower the probability of making a Type II error. In this research, it means the probability of detecting the DNA pattern effect is higher. Experimental results with low statistical power will lead to conclusions that are invalid. Therefore, a minimum level of statistical power must be taken into consideration when designing experiments. Whilst designing experiments, it is common to have a statistical power of 80 percent or greater. This means a 20 percent chance of making a Type II error. This is different to the 5 percent chance of making a Type I error for the standard value for the significance level, that is, 0.05 (Becker, 2011, Pg 52-54). Therefore, given the effect size, statistical power, and significance level, the sample size can be determined. In this research, to conform to the hypothesis in section 3.3.2, the values of effect size, significance level, and statistical power were chosen to be 0.99, 0.01, and 0.99. This means, to estimate a sample size in order to at least detect DNA pattern effect of 0.99, with an 99 percent chance of detecting DNA pattern effect if it is true, that is, 1 percent chance of making Type II error and 1 percent chance of detecting DNA pattern effect if there is no such effect, that is, making Type I error.

The sample size was calculated using the statsmodels TTestIndPower library (Statmodels,

2019) and programmed in python 3 as shown in listing 3.1.

Listing 3.1 Estimating sample size

```
from statsmodels.stats.power import TTestIndPower
#parameters for power analysis
effect = 0.99
alpha = 0.01
power = 0.99

#Estimate sample size
analysis = TTestIndPower()
result = analysis.solve_power(effect,power=power,
nobs1=None,ratio=1.0,alpha=alpha)
print('Sample Size: %.3f' % result)
```

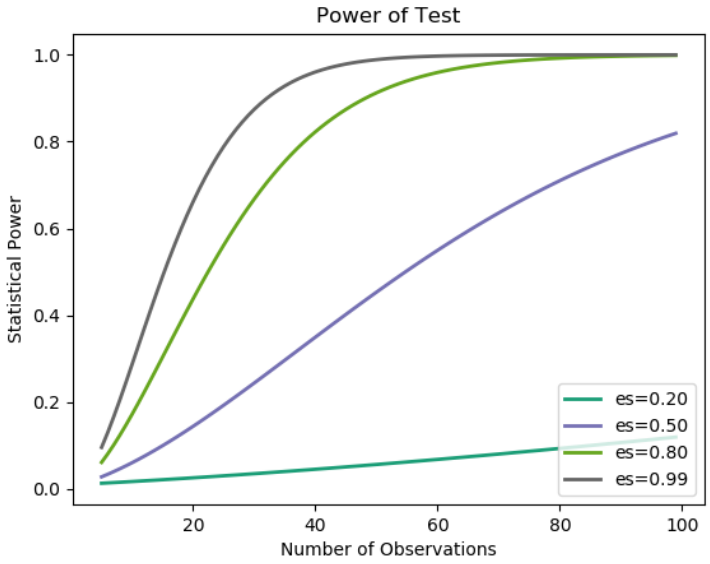


Fig. 3.3 Power of Test

Given, the values of effect size, alpha, and power to the TTestIndPower along with its solve_power class it gives the value for the estimated sample size. For this study, the estimated sample size was 50.733. Therefore, 50 samples was chosen as the sample size for this research. Since data will be collected from ten machines; each machine, will have 5 trails consisting of 50 samples in each trial. The plot between statistical power and number of observations, that is, sample size is plotted as seen in figure 3.3.

3.3.5 Data Collection and Pre-Processing

The data was collected from ten computers and later stored in ten USB 2.0 devices with 128 MB storage capacity. Each of the ten computers had the same specification, Windows 10 OS, 16 GB RAM, Intel i7-9700 (3 GHz, 8 cores) processor. The data collection was divided into control group and experimental group. The control group contains three memory forensic acquisition tools, FTK Imager, Magnet RAM Capturer, and Belkasoft RAM Capture, and one non-forensic tool TankiOnline (Tanki, 2018). The reason for opting for the non-forensic tool TankiOnline was it is a free computer game software which can be run on a Windows 10 machine, and to see if any of the ML classifiers would be able to differentiate between this game software and a forensic tool(s) in both the groups when executed on a Windows 10 machine.

The experimental group consists of the three aforementioned forensic tools only. In the control group, the forensic tools and the non-forensic tool is executed but are not imaging memory, and hence there is no memory acquisition taking place. Whereas, in the experimental group the forensic tools are executed to acquire memory from the machines. As for the order of execution, firstly the forensic tools in the experimental group were executed in the order of Magnet, FTK, and Belkasoft as shown in listing 3.2. Then, after the experimental group data has been collected, the forensic tools in the control group were executed in the same order as experimental group, followed by the execution of the non-forensic tool TankiOnline. The data was collected in the same order as mentioned previously on all ten machines by inserting a USB device, which contained the data collection script for each machine. Therefore, the experiment is being repeated exactly the same manner on all the ten machines. Whilst the tools were executed on the machines, the MIOC parameters concerning to the tools in the experimental group and control group were collected. The data collection procedure is shown in the figure 3.4.

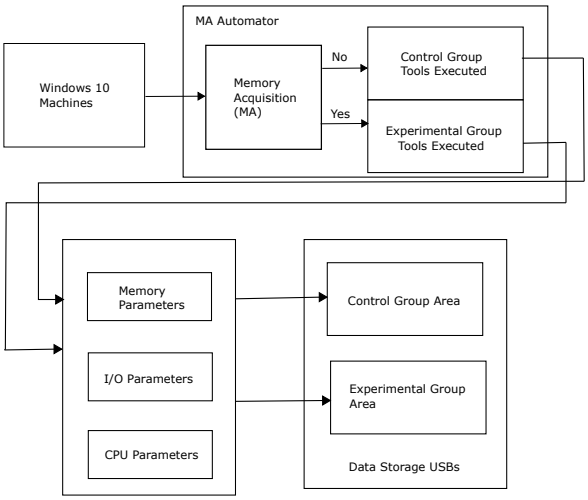


Fig. 3.4 Data Collection Procedure

The MIOC parameters are gathered with the help of an USB device. It is a good practice guideline to image the physical memory using an USB device on which a forensic acquisition tool is present Williams (2012). When the USB device is inserted into the USB port of the computer, the memory acquisition (MA) automator block is executed. In the MA automator, the forensic tools and a non-forensic tool automatically get executed. Now, the control group tools get executed if memory is not being acquired. Or else, if memory is being acquired then experimental group tools get executed. Python 3 scripts were written to execute the tools in both groups. Part of the data collection (DC) code for the experimental group is shown in listing 3.2.

Listing 3.2 DC Python Code for Experimental Group

```
def expgrp():
...
    for j in range(k):
        print(array[j])
        if(array[j]=="Magnet"):
                procname = "MagnetRAMCapture"
                MagCap(vol)
        elif(array[j]=="Ftk"):
                procname = "FTK"
                FtkCap(vol)
        elif(array[j]=="Belkasoft"):
                procname = "RamCapture64"
                BelkaCap(vol)
        ...
```

The explanation for the code in listing 3.2 is as follows: when the exgrp() function is called, the for loop iterates over the elements in the array. The array contains the list of forensic tools namely FTK Imager, Magnet, and Belkasoft. In total there will be three separate iterations in the experimental group. For example, during the first iteration, the Magnet tool will be run, which is invoked by calling the autoexecution function, MagCap(vol). Then, FtkCap(vol) and BelkaCap(vol) will be called respectively as shown in listing 3.2.

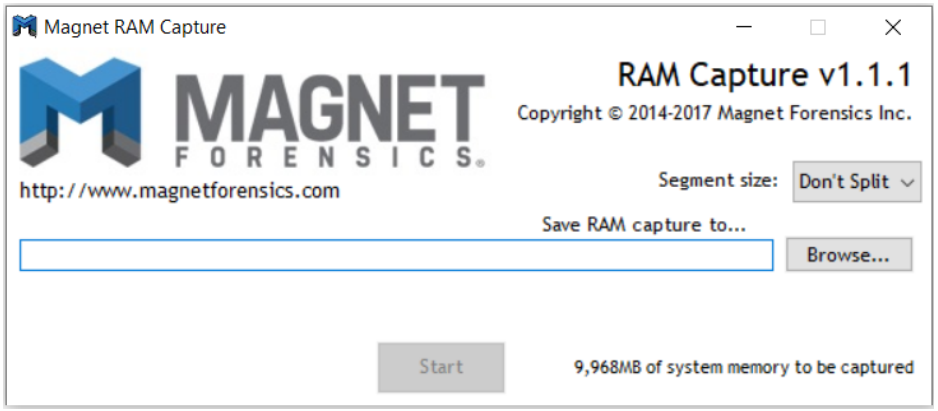


Fig. 3.5 Autoexecution of Magnet Forensic Software

The purpose of the autoexecution function is to automatically execute the forensic software to acquire memory. In this case, as shown in figure 3.5, the path where the memory file is to be stored is automatically selected by clicking on the browse button and then entering the path. Then, the start button is clicked to capture memory without the user's intevention. The part of python code for Magnet's autoexecution function is shown in listing 3.3.

Listing 3.3 Autoexecution code for Magnet

```
def Magcap(vol):
        pyauto = Dispatch("AutoItX3.Control")
...
        #click browse
        pyauto.WinActivate("Magnet␣RAM␣Capture", "");
        pyauto.sleep(3000);
        #ControlClick ( "title", "text", controlID ,
         button , clicks , x , y)
        pyauto.ControlClick ("Magnet␣RAM␣Capture","",
         3,"primary", 1 , 36 , 13);
        ...
        #Click again on start to mem capture
        pyauto.ControlClick ("Magnet␣RAM␣Capture",
        "Start", 5,"left", 1 ,44, 16);
...
```

The autoexecution function makes use of AutoItX (Xu, 2016), which is an automation framework for Windows operating system. Then the button co-ordinates are entered as parameters to the ControlClick function which is used to click on browse button to enter the memory image path, and then click on start button for forensic memory acquisition.
Once the forensic tool is executed and memory is being acquired, the MIOC parameters are collected for that particular forensic tool. The code snippet for MIOC parameters collection is shown in listing 3.4.

Listing 3.4 MIOC Parameters Collection

```
...
if procname in p.info['name']:
##Collect the memory parameters for magnet process
        list1 = p.memory_full_info()[2:]
##collect I/O parameters
        list2 = p.io_counters();
#non-blocking
       m = (p.cpu_percent(interval=None)/psutil.cpu_count())
#blocking
        n = (p.cpu_percent(interval=1)/psutil.cpu_count())
##collect cpu parameters
        list3 =[m,n]
##collect cpu parameters
        list4 = p.cpu_times()[0:2]
...
```

The collection of MIOC framework is facilitated by a python framework known as psutil Rodola (2019). It provides functions such as *memory_full_info*, *io_counters*, *cpu_percent*, and *cpu_parameters* to gather MIOC parameters. The collection for each forensic tool were split into five trials. Each trial in turn has fifty MIOC parameters collected with one second difference between each MIOC parameter collection. The one second difference were chosen because in 3.3.4, the sample size was determined to be 50, and the total time to make a memory image for a Windows 10 OS with 16 GB of physical memory and Intel i7-9700 (3GHz, 8 core) processor is approximately 120 seconds. Therefore, each trial contains 50 samples, that is, 41 percent ($50/120 = 0.41$) of the memory data acquired. In other words, this research intended to detect forensic memory acquistion within 50 seconds (less than a minute) from its inception. During acquisition, the MIOC parameters were stored in excel sheets because in this research python 3 libraries such as Pandas (Pan, 2020), which supports data processing to and from the excel sheets. Data gathered from each computer were stored in the respective USB device for data formatting purposes.

Data Pre-processing

After MIOC parameters were stored into USB devices, data is processed into a format which can be inputted to ML classifiers. Figure 3.6 shows the approach taken to format data. As

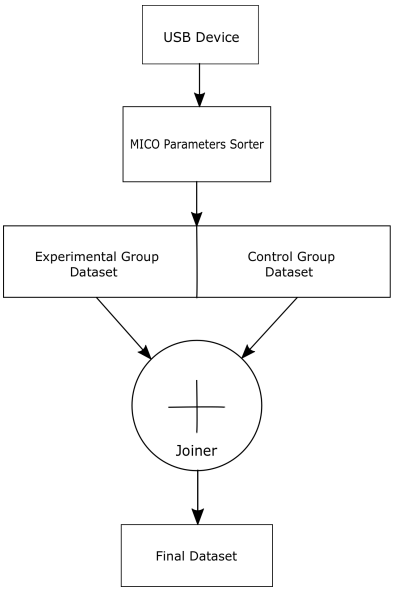


Fig. 3.6 Data Formatting Steps

already seen in section 3.3.5, the data is ogranised into experimental group and control group in each USB device. The structure of a USB device as shown in figure 3.7. The data from each USB is copied on to a computer for further processing. The USB folder contains two main folders, 'Experimental Group' and 'Control Group' respectively. The Experimental Group contain three forensic tool folders namely Belkasoft, Ftk, and Magnet, and each of these forensic tool folders contain the trial folder containing of fifty excel sheets namely writing1.xls to writing50.xls. It is in these excel sheets the MIOC parameters are stored. For example, writing1.xls contains MIOC parameters when the Belkasoft forensic tool was acquiring memory as shown in figure 3.8.

In the second step, the data from the $'USB/ExperimentalGroup/Belkasoft/Trial1/'$ location is read in a loop. This loop iterates over the fifty excel sheets, five Trial folders, and three forensic tools in the Experimental Group Folder and stores the sorted MIOC parameters

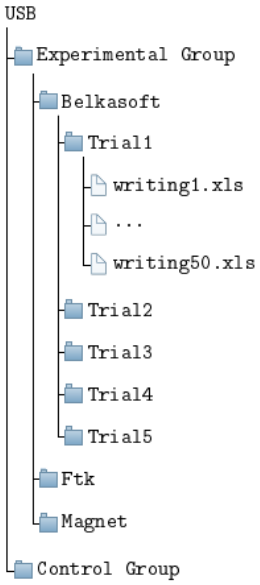


Fig. 3.7 USB Structure

	memory_info		IO_Counters			cpu_percent			cpu_times	
num_page	11540		read_coun	16909		non-blocki	0		user	0.671875
peak_wset	38789120		write_cour	14345		blocking	3.7125		kernel	2.203125
wset	37203968		read_bytes	9.4E+08						
peak_page	494896		write_bytes	9.4E+08						
paged_poo	416880		read_time	4959						
peak_nonp	43040		write_time	248808						
nonpaged_	36144									
pagefile	11014144									
peak_page	11862016									
private	11014144									
USS	24133632									

Fig. 3.8 MIOC Parameters of writing1.xls

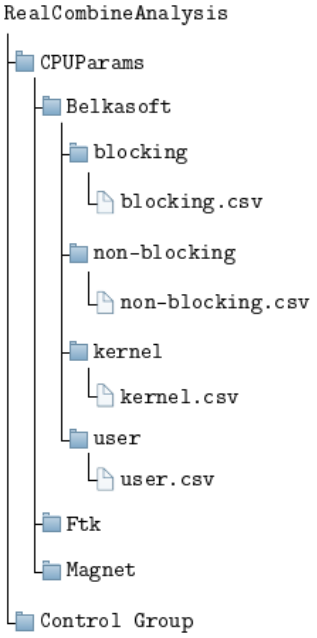
in the folder *'RealCombineAnalysis/CPU_Params/'*. For example, the CPU parameters are sorted as shown in figure 3.9a. A python snippet showing the implementation of the aforementioned steps is shown in listing 3.5.

Listing 3.5 MIOC Parameters Arrangement

```
...
for t in range(0, len(tool_list)):
        for k in range(0, len(memparam_list)):
                for j in range(1,6):
                        for i in range(1,51):
                                df = pd.read_excel
                                (path_to_usb_expgrp)
                                df.to_csv(path_to_file_loc)
...
```

It consists of four for loops with their respectively variables *t*, *k*, *j*, and *i*. The variable *t* iterates over the forensic tool list. This would be three for experimental group as it consists of three forensic tools. The second variable *k* iterates over the parameters that were captured for the forensic tools whilst they were acquiring memory in case of experimental group and not acquiring memory in case of control group. The value of *k* depends on which MIOC parameter is being addressed. For example, for CPU parameters the value of *k* is four since it consists of four parameters. Therefore, in listing 3.5, the for loop iterates over the list for four times. The next variable *j* iterates over the Trial folders as shown in figure 3.7. The final parameter *i* iterates over fifty excel sheets present in the Trial folder.

For example, after the individual CPU parameter has been sorted as shown in figure 3.9, it is further sorted in such a way that the four CPU parameters form the features or distinctive native attributes (DNA) for both the experimental group and the control group.The four DNAs are arranged in four columns, and the fifth column is attributed to the class name. Now, the datasets of experimental group and control group as shown in figure 3.10 are joined to form the final dataset. These datasets were used as inputs to various ML agorithms as discussed in Chapter 4. In Chapter 5, these datasets were transformed to images and then analysed using 3L-CNNs.



(a) CPU Parameter arrangment

0	13:26:18	12.5
1	13:26:19	12.5
2	13:26:20	12.5
3	13:26:21	9.575
4	13:26:22	6.6375
5	13:26:23	6.6375
6	13:26:24	6.25
7	13:26:25	9.175
8	13:26:26	8.7875
9	13:26:27	7.8125
10	13:26:28	7.6125
11	13:26:29	6.25
12	13:26:30	6.8375

(b) Blocking CPU parameter of blocking.xls

Fig. 3.9 Folder Structure of CPU Parameters

blocking	kernel	nonblocking	user	class
12.7	1.6875	0	0.203125	FTK-AM
12.5	2.703125	12.525	0.265625	FTK-AM
12.3	3.734375	12.4	0.28125	FTK-AM
12.7	4.65625	6.3	0.390625	FTK-AM
12.7	5.671875	12.6	0.421875	FTK-AM
12.5	6.6875	12.4	0.46875	FTK-AM
12.5	7.671875	12.6	0.515625	FTK-AM
13.475	8.625	12.2125	0.65625	FTK-AM
12.7	9.625	12.6	0.703125	FTK-AM
12.7	10.65625	12.6	0.71875	FTK-AM
12.7	11.67188	12.6	0.75	FTK-AM
11.1375	12.59375	12.525	0.796875	FTK-AM
11.1375	13.51563	12.2125	0.796875	FTK-AM

(a) Experimental Group Dataset

blocking	kernel	nonblocking	user	class
2.3125	0.140625	0	0.046875	FTK-NAM
0	0.140625	0	0.046875	FTK-NAM
0	0.140625	0	0.046875	FTK-NAM
0	0.140625	0	0.046875	FTK-NAM
0	0.140625	0	0.046875	FTK-NAM
0	0.140625	0	0.046875	FTK-NAM
0	0.140625	0	0.046875	FTK-NAM
0	0.140625	0	0.046875	FTK-NAM
0	0.140625	0	0.046875	FTK-NAM
0	0.140625	0	0.046875	FTK-NAM
0	0.140625	0	0.046875	FTK-NAM
0	0.140625	0	0.046875	FTK-NAM
0	0.140625	0	0.046875	FTK-NAM
0	0.140625	0	0.046875	FTK-NAM

(b) Control Group Dataset

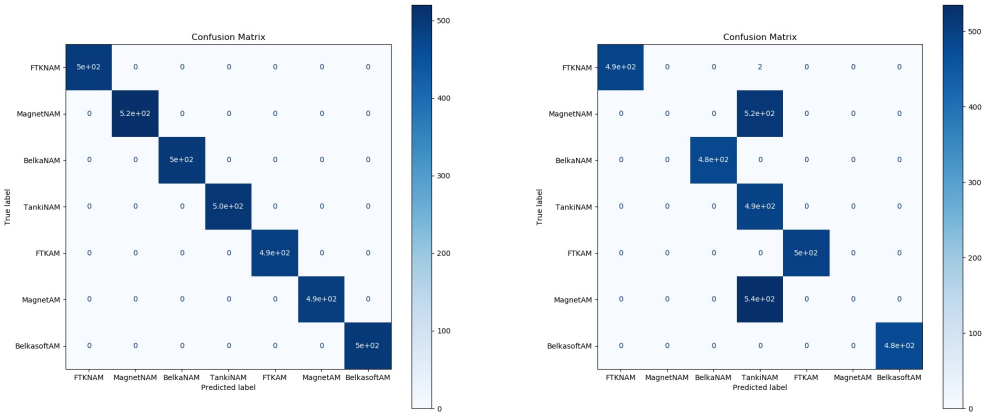
Fig. 3.10 CPU datasets for Experimental and Control Group

3.3.6 Model Evaluation and Interpretation

In this research, the metrics used for evaluating and interpreting the machine learning models are the confusion matrix, true positive rate (TPR) and false positive rate (FPR) and triangulating these results (Heale and Forbes, 2013) with area under the receiver operating characteristic (ROC) curve, precision-recall curves, learning curves, and bias-variance trade-off, and these terms are explained in the following sections.

Confusion Matrix

A confusion matrix (CM) summarises the performance of a classification algorithm. In this research since multiclassification performance is measured, evaluating accuracy alone can be misleading because classification accuracy hides details that are used to understand a model's performance (Hossin and Sulaiman, 2015). For example, when the data has more than 2 classes, a classification accuracy of 90 percent or more could be encountered, but what can be hidden is whether all classes are predicted well or are there any classes that has been ignored by the model. Therefore, interpreting a confusion matrix can give a better idea of a classification model's performance. The library package sklearn (sklearnCM, 2019) was used to generate confusion matrices in this study. There are two cases for a confusion matrix, an ideal and non-ideal case as shown in figure 3.11a and 3.11b respectively.



(a) Ideal CM (b) non-ideal CM

Fig. 3.11 Confusion Matrix

Consider the ideal CM in figure 3.11a, the colour bar is an indicator of the magnitude of each and every element in the CM. Darker the colour higher the magnitude of an element in the CM, and lighter the colour lesser the magnitude of an element. For an ideal CM, its diagonal elements will have darker colour. This is because the diagonal of a CM represents true positives, and the columns represents false negatives, and the rows represent false positives. Whereas, in the non-ideal case in figure 3.11b, at least one of the diagonal elements in the CM will not have a dark colour, and a row or column element will have a darker colour indicating that misclassification has taken place. The accuracy, precision and recall can be calculated from the confustion matrix (sklearnCM, 2019).

Accuracy

Accuracy is the most widely used evaluation metric for classification performance. It is defined as the ratio between correctly classified samples ($TP + TN$) to total number of

positive and negative samples.

$$Acc = \frac{TP+TN}{TP+TN+FP+FN}$$

The error rate can be calculated from accuracy as, $ERR = 1 - Acc = (FP+FN)/(TP+TN+FP+FN)$. The error rate which is also known as the misclassification rate gives the number of misclassified samples from both positive and negative cases. The drawback with accuracy is two classifiers can have the same accuracy yet have incorrect predictions (Hossin and Sulaiman, 2015).

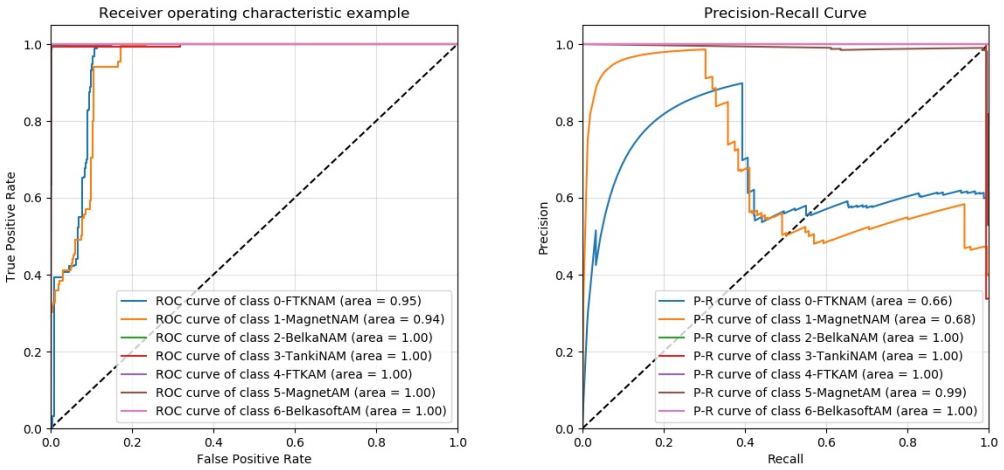
Sensitivity or True Positive Rate (TPR)

Sensitivity, TPR, or recall is the number of correctly classified positive samples to the total number of positive samples. On the other hand, specificity or true negative rate (TNR), which is the the inverse of recall is the ratio of negative samples that are correctly classified to the total number of negative samples. Therefore, specificity is the proportion of negative samples that were classified correctly, and sensitivity is the proportion of positive samples that were classified correctly. Sensitivity is dependent on TPs and FNs which are present in the same row of the confusion matrix (Hossin and Sulaiman, 2015). Similarly, specificity depends on TNs and FPs. The formula for sensitivity or recall is

$$TPR = \frac{TP}{TP+FN}$$

Positive and Negative Predictive Value

Positive predictive value (PPV) and negative predictive value (NPV) is the reflection of the prediction's performance. PPV or precision is the proportion of positive samples that have been classified correctly to the total number of positive predicted samples as shown in equation 3.4. Whereas, the NPV which is also known as inverse precision or true negative



(a) ROC Curve (b) P-R Curve

Fig. 3.12 ROC and P-R Curves

accuracy is the measure of the proportion of negative samples that have been classified correctly to the total number of samples that were negatively predicted. False discovery rate (FDR) and False omission rate (FOR) metrics are inverses of PPV and NPV respectively (Hossin and Sulaiman, 2015).

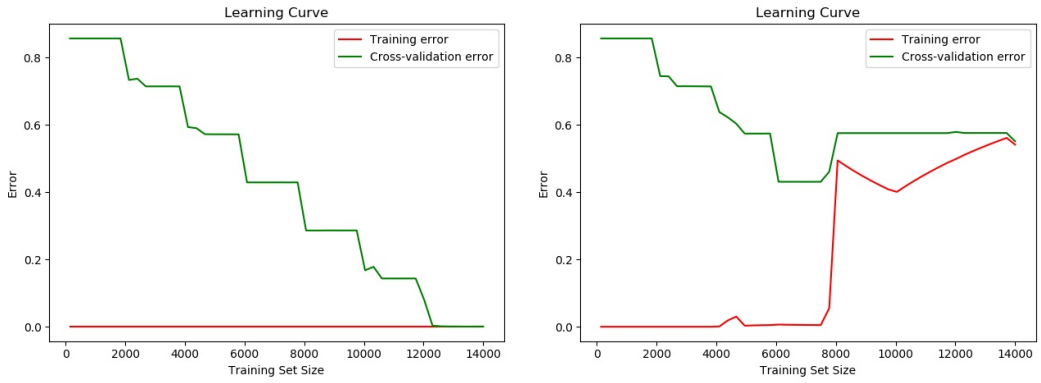
$$PPVorPrecision = \frac{TP}{FP+TP} = 1 - FDR \tag{3.4}$$

ROC and PR curves

The area under the Receiver Operating Characteristic (ROC) curve and Precision-Recall curve, in a multiclassification setting will show how each class has performed. In this research, it is used to cross-validate the results of the confusion matrix. ROC curve is a plot between true positive rate and false positive rate, whereas Precision-Recall (P-R) curve is a plot between true positive rate and positive predictive value. In the example figure 3.12, the ROC and P-R curve show that class 0 and class 1 are the most effect when compared to other classes for that particular model (Hossin and Sulaiman, 2015).

Learning Curves and Bias-Variance Tradeoff

Learning curves tell us that whether a model has overfit, underfit, or is a good fit to the data. The learning curves are shown in figure 3.13. The learning curve is a plot between training sample size and, training and cross-validation error.



(a) Good fit to the data (b) High bias

Fig. 3.13 Learning Curves

As the training size increases, if the training and cross- validation error decreases then it means that the model has learnt the relevant characteristics from the data as shown in figure 3.13a. Another case that is most likely to occur is as the training size increases, the training and validation errors do not decrease and ends up converging at point which has some amount of significant error as shown in figure 3.13b, which means the model exhibits bias and has underfit the data. If the curves diverge from each other it is said to exhibit variance and will overfit the data. The formula used to measure the bias and variance in a model is:

$$Error = bias^2 + variance + noise \tag{3.5}$$

The third term in equation 3.5, noise, is also known as the irreducible error and it cannot be controlled in an experiment. The first term, square of the bias is the average error in exhibited in the model, and the second term variance tells us how much the input varies depending on

the training set. It is possible to change bias and variance in an experiment, but an important point to note is that decreasing bias will increase the variance and vice-versa. Therefore, one of the goals in an experiment is to find a model that exhibits optimal bias and variance, (Stephen, 2014).

3.3.7 Supporting and Modifying Hypothesis

In this research, the hypothesis was supported, modified, or rejected by obtaining the results and comparing them to the threshold values. The criteria for choosing threshold values is described below: In section 3.2, the step number five and six are iterable. This is one of the important phases of the scientific method because it allows the modification of hypothesis to fit with the observations provided the required consistency is achieved. In chapter 4, this consistency is achieved by observing the results of the K-fold cross-validation, and in chapter 5, the number of epochs. The threshold value was determined from the observations of the ML classifiers performance on the memory parameters dataset as shown in table 3.1. This dataset was chosen as it consists of higher number of features (10) when compared to I/O dataset (6 features), and CPU dataset (4 features). The size of the memory parameters dataset is (17500,11), that is, 17500 rows, the first 10 coloumns consists of features, and the 11th coloumn consists of the 7 classes. There are 3500 iterations for each class, hence the 17500 rows. (Bitbucket, 2020d).

ML Classifier	Accuracy	Precision	Recall	F-Measure	Detection Error Rate
ADA Boost	84.28	31.48	45.65	37.26	73.4
Decision Tree	99.99	99.99	99.99	99.99	<5
Gaussian NB	96.74	89.31	88.4	88.85	18.4
Gradient Boost	99.99	99.98	99.98	99.98	<5
KNeighbours	99.99	99.97	99.97	99.97	<5
LDA	97.4	91.25	90.85	91.04	17.8
Random Forest	99.99	99.98	99.98	99.98	<5
SVM	91.68	62.88	49.78	55.56	40.0

Table 3.1 Threshold Values

From table 3.1, three ML classifiers have their detection error rate less than 5 percent. This was the first criteria and threshold value that was chosen because, the detection error rate was calculated from the bias-variance trade-off rather than complementing the accuracy, that is, (1-accuracy). The accuracies for the four ML classifiers are equal, but their precision and recall vary. In this group, KNeighbours has the least precision, recall, and F-measure and will qualify for the threshold level setting. Hence, the rules this study proposes to determine the threshold level are:

1. Select the ML classifiers with least detection error rate.

2. The ML classifier with least accuracy, precision, recall, and F-measure in that group will be set as threshold values or levels. That is, if any ML classifier which is equal or greater than the threshold level will support the hypothesis. Likewise, any ML classifier below the threshold level will not support the hypothesis. In which case, the hypothesis will be modified or rejected based on the consistency of the results achieved.

This study does not use the statistical hypothesis testing to determine whether there is a significant difference between two groups because it will not satisfy the aforementioned threshold level criteria, and may lead to misleading conclusions (Jason, 2016). For example, if a t-test is used to find if there is difference in means between KNeighbours and LDA, the result will fail to reject the null hypothesis as the p-value (0.016) is less than the significance level (α) of 0.05 which means there is no difference between the two classifiers, but there is a difference between KNeighbours and LDA based on the threshold values in table 3.1.

3.4 Conclusion

In this chapter the research method in AI and the scientific method were discussed. Then a research method for this study had been proposed and justified after being informed by the research approach in AI and the scientific method. The experiment had been designed to gather and pre-process the data. Various evaluation techniques have been mentioned that were used in chapter 4 and chapter 5 to evaluate a model. Finally, the threshold values selection are justified based on which the hypothesis is supported, modified or rejected.

Chapter 4

Detecting Forensic Memory Acquisition using Machine Learning Algorithms

4.1 Introduction

In this chapter, the classification results of eight machine learning models are evaluated for performance. These eight algorithms were chosen as it was found to be extensively used in the literature in the field of cyber security and forensics as seen in section 2.5.1. The metrics used for evaluating the machine learning models are true positive rate (TPR) and false positive rate (FPR) which can be derived from the confusion matrix, and cross-checking these results with the area under the receiver operating characteristic (ROC) curve, P-R Curve, learning curve, and detection error rate of a machine learning model.

4.2 Detecting Memory Acquisition using Memory Parameters

In this section, the results of eight machine learning classifiers are analyzed to gather evidence to support or reject the memory detection hypothesis. The machine learning classifiers will be analyzed by metrics mentioned in section 3.3.6 by using confusion matrix, and various curves to determine which ML classifier model fits the data well and which model does

not fit the data well. Therefore this will result in wither falsifying, accepting, or modifying the hypothesis. Since the classifiers will be used against memory parameters, the dataset is explained briefly. Only a part of the memory parameter dataset is shown in figure 4.1. The complete dataset can be found at (Bitbucket, 2020d) .

nonpaged_pool	numpage	paged_pool	pagefile	peak_nonpaged_pool	peak_paged_pool	peak_pagefile	peak_wset	USS	wset	class
7200	905	161848	2072576	7200	161848	2072576	3477504	978944	3477504	FTK-NAM
34864	11896	697696	16474112	35136	700112	23973888	47022080	28487680	47022080	FTK-NAM
34864	11896	697696	16474112	35136	700112	23973888	47022080	28487680	47022080	FTK-NAM
34864	11896	697696	16474112	35136	700112	23973888	47022080	28487680	47022080	FTK-NAM
34864	11896	697696	16474112	35136	700112	23973888	47022080	28487680	47022080	FTK-NAM
34864	11896	697696	16474112	35136	700112	23973888	47022080	28487680	47022080	FTK-NAM
34864	11896	697696	16474112	35136	700112	23973888	47022080	28487680	47022080	FTK-NAM
34864	11896	697696	16474112	35136	700112	23973888	47022080	28487680	47022080	FTK-NAM
34864	11896	697696	16474112	35136	700112	23973888	47022080	28487680	47022080	FTK-NAM
34864	11896	697696	16474112	35136	700112	23973888	47022080	28487680	47022080	FTK-NAM
34864	11896	697696	16474112	35136	700112	23973888	47022080	28487680	47022080	FTK-NAM

Fig. 4.1 Memory Dataset

The dataset consists of 17500 rows in total and 11 columns. Therefore, the size of dataset is $(17500, 11)$. The first 10,000 rows relate to the iterations when the forensic tools were executed but memory was not being acquired. And, the following rows consists of 7500 iterations when memory was being acquired by a forensic tool. During each iteration certain memory parameters were gathered. These parameters, also known as features, are defined in the following table 4.1

Memory Parameter	Description
Non-paged Pool	Amount of memory a process is using that cannot be moved out to the pagefile and remains in memory.
Numpage	The number of pagefaults
Paged_Pool	Amount of memory a process is using in pageable memory region.
Pagefile	The data that is not stored in the physical memory
Peak_nonpaged_pool	Maximum value of nonpaged pool in bytes
Peak_paged_pool	Maximum value of paged pool in bytes
Peak_pagefile	Maximum value of pagefile
Peak_wset	Peak value of Wset
Unique set size (USS)	Memory that is freed when a process terminates. Every process has its own USS.
Wset	The non-swapped memory a process has utilised. It is also known as resident set size.

Table 4.1 Memory Parameters

AdaBoost Classifier

The model is evaluated based on confusion matrix (Mem, 2020b), and then cross-checked with ROC curve (Mem, 2020d), P-R curve (Mem, 2020c), and bias-variance trade-off (Mem, 2020a). The confusion matrix clearly distinguishes between FPs and FNs as shown in figure 4.2. From confusion matrix it is observed that classes 0, 2, and 6 are severly effected as they have 0 TPs, and this in turn affects the performance of other classes, for example, even though class 1 has been classified around 472 of it instances, but it has misclassified around 552 instances. Similarly, class 4 has correctly identified around 492 classifiers, but it has misclassified about 1031 instances. It appears only class 3 and 5 have fit the data well.

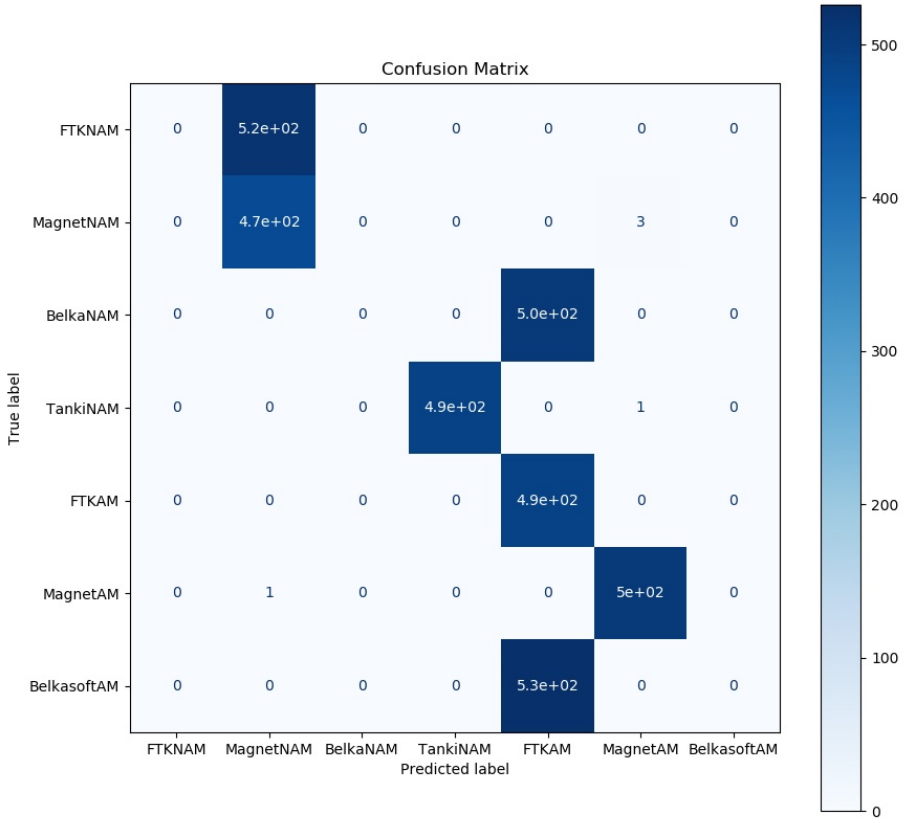
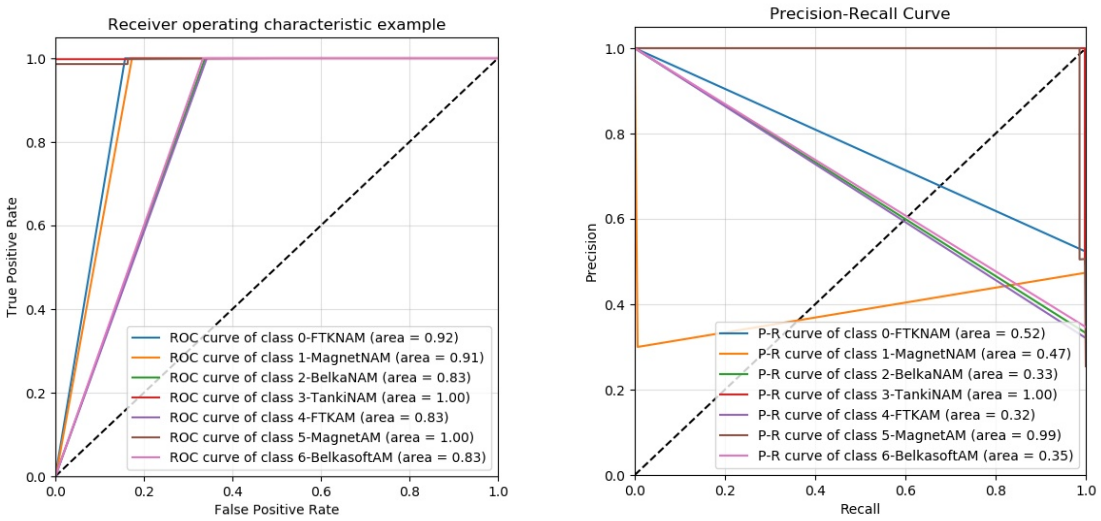


Fig. 4.2 Confusion Matrix for AdaBoost Classifier

To further support the results from the confusion matrix, the results of ROC curve in figure 4.3a shows that except for classes 3 and 5, the other classes are below the threshold level. Also the P-R curve have low values for classes other than 3 and 5, which indicates these classes have either high FPs or FNs, and have been misclassified as confirmed from the results of the confusion matrix. Also, classes 0, 2, 4 and 6 have high recall, but low precision. This means, the model is able to identify all instances of those classes, but out of those instances which were identified, only around 30 to 40 percent of them have been correctly detected. And, these poor results are either due to higher number of FPs and FNs as seen in

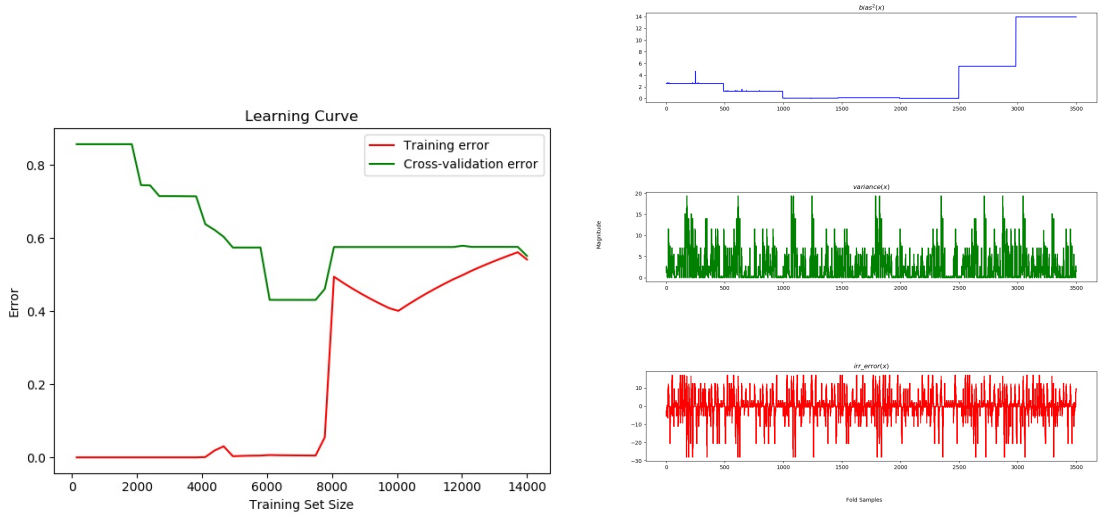


(a) ROC curve for AdaBoost (b) Precision-Recall Curve for AdaBoost

Fig. 4.3 ROC and P-R Curves for AdaBoost

the confusion matrix in fig 4.2. Therefore, the results of AdaBoost classifier model is poor at detecting memory acquisition, and hence falsifies the hypothesis. As for the bias-variance tradeoff, the learning curve for AdaBoost classifier as shown in figure 4.4a, and it is seen that it has high variance as the training error is lesser than the validation error, and high bias as both the curves converge at a point higher than the threshold value. Therefore, increasing the number of samples will not improve the model's performance. Hence, this model is not suited to detect forensic memory acquisition.

And the bias-variance decomposition plots in figure 4.4b show bias and variance are above threshold values. This is the reason why AdaBoost classifier has higher number of FPs and FNs, and lower precision. The detection error rate (Δ_r) is calculated as the sum of bias and variance. For AdaBoost, Δ_r is 73.46 percent which is high. Therefore, AdaBoost is not good at detecting forensic memory acquisition using memory parameters.



(a) Learning Curve for AdaBoost (b) Bias-Variance Decomposition for AdaBoost

Fig. 4.4 Learning and B-V Curves for AdaBoost

GaussianNB

The confusion matrix clearly distinguishes between FPs and FNs as shown in figure 4.5. From confusion matrix it is observed that classes 0, and 1 are severly effected as class 0 has 280 FPs and 86 FNs, whereas other classes for this model have performed well. It appears only class 0 and 1 do not fit the data well.

To further support the results from the confusion matrix in figure 4.5, the results of ROC curve (Gau, 2020d) in figure 4.6a shows that except for classes 0 and 1, the other classes perform well. Also the P-R curve in figure 4.6b have low values for classes 0 and 1, which indicates these classes have either high FPs or FNs, and have been misclassified as confirmed from the results of the confusion matrix. These poor results are either due to higher number of FPs and FNs as seen in the confusion matrix in figure 4.5. Therefore, the results of GaussianNB classifier model is poor at detecting memory acquisition, and hence falsifies the hypothesis. As for the bias-variance tradeoff in figure 4.7, the learning curve for GaussianNB classifier as shown in figure 4.7a, and it is seen that the variances decreases as the training

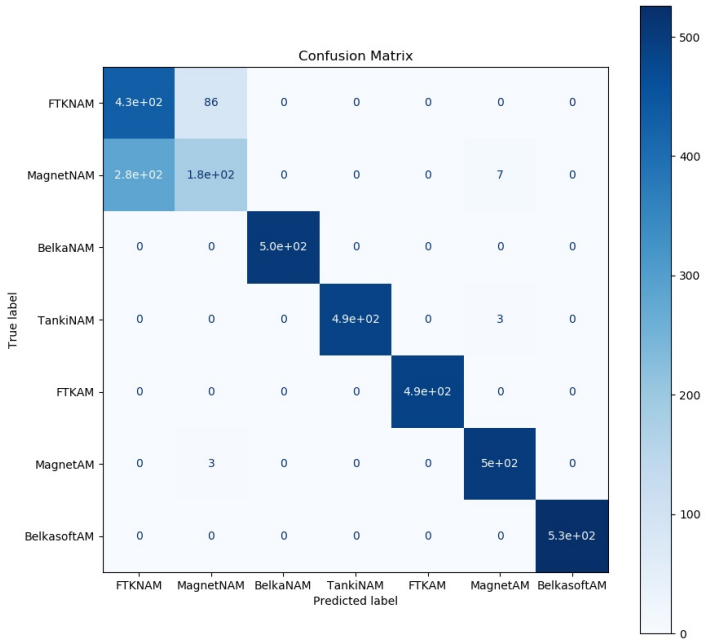
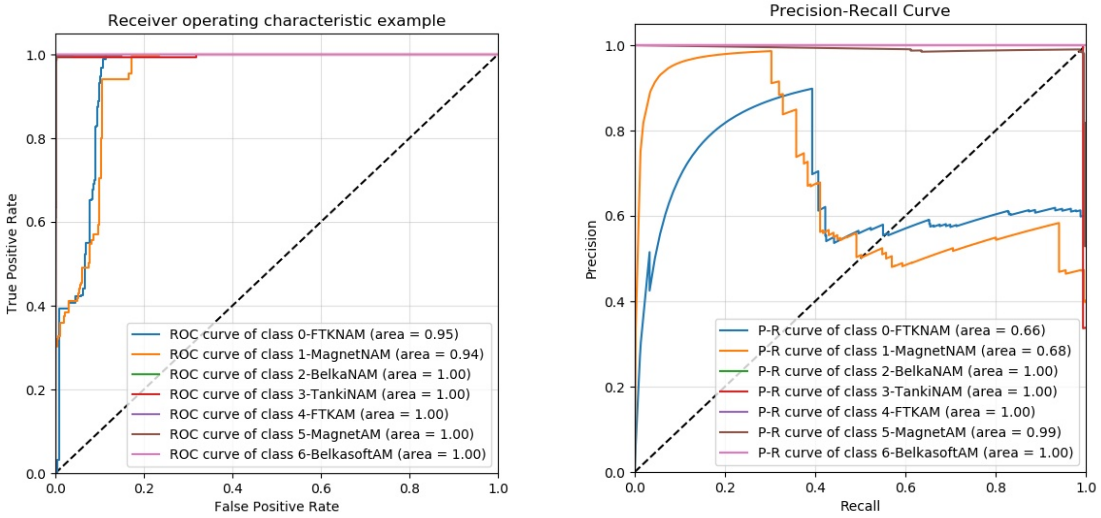
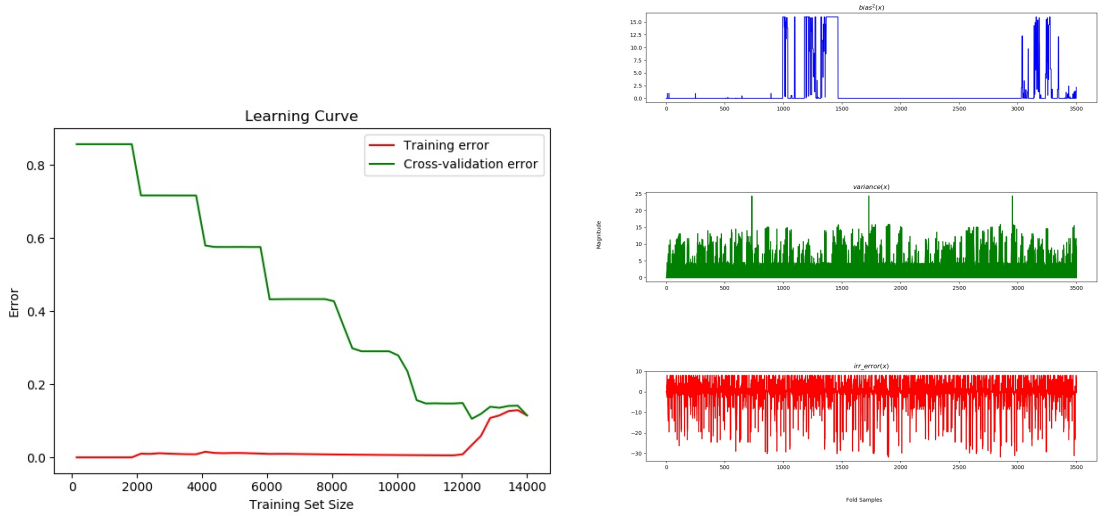


Fig. 4.5 GaussianNB confusion matrix



(a) ROC curve for GaussianNB (b) Precision-Recall Curve for GaussianNB

Fig. 4.6 ROC and P-R Curves for GaussianNB



(a) Learning Curve for GaussianNB (b) Bias-Variance Decomposition for AdaBoost

Fig. 4.7 Learning and B-V Curves for AdaBoost

size increases, but the both the curves converge at a point greater than the threshold value indicating that this model exhibits bias.

And the bias-variance decomposition plots in figure 4.7b show bias and variance are above threshold values. This is the reason why GaussianNB classifier has of FPs and FNs, and lower precision and recall. The detection error rate (Δ_r) is calculated as the sum of bias and variance. For GaussianNB, Δ_r is 18.4 percent. Therefore, GaussianNB is not good at detecting forensic memory acquisition using memory parameters. LDA classifier also shows similar performance to GaussianNB and has a Δ_r value of 17.8 percent. LDA also does not perform well in detecting memory acquisition.

SVM

The confusion matrix clearly distinguishes between FPs and FNs as shown in figure 4.8. From confusion matrix it is observed that classes 0, and 1 are severly effected as class 0 has zero TPs and has 520 FNs, and class 1 has 1023 FPs. This means class 1 has misclassfied

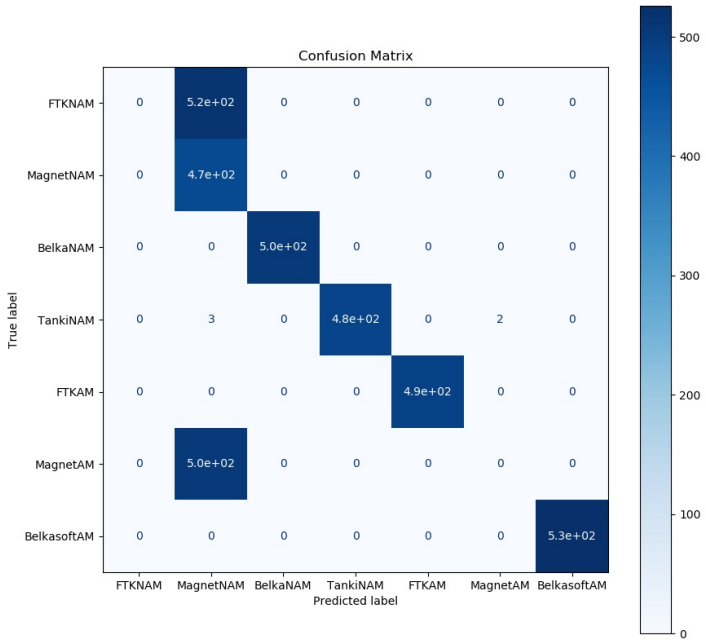
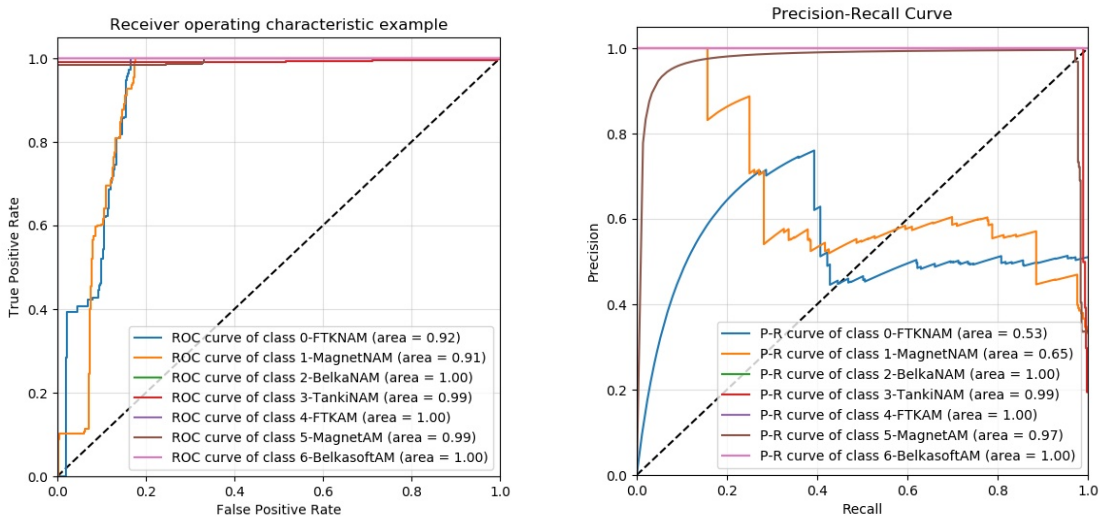


Fig. 4.8 SVM confusion matrix

520 instances of class 0 and 500 instances of class 5. The other classes for this model have performed well. It appears only class 0 and 1 do not fit the data well.

To further support the results from the confusion matrix in figure 4.8, the results of ROC curve in figure 4.9a shows that except for classes 0 and 1, the other classes perform well. Also the P-R curve in figure 4.9b have low values for classes 0 and 1, which indicates these classes have either high FPs or FNs, and have been misclassified as confirmed from the results of the confusion matrix. These poor results are either due to higher number of FPs and FNs as seen in the confusion matrix in figure 4.8. Therefore, the results of SVM classifier model is poor at detecting memory acquisition, and hence falsifies the hypothesis. As for the bias-variance tradeoff in figure 4.10, the learning curve for AdaBoost classifier as shown in figure 4.10a, and it is seen that it has high variance as the training error is lesser than the validation error, and high bias as both the curves converge at a point higher than the



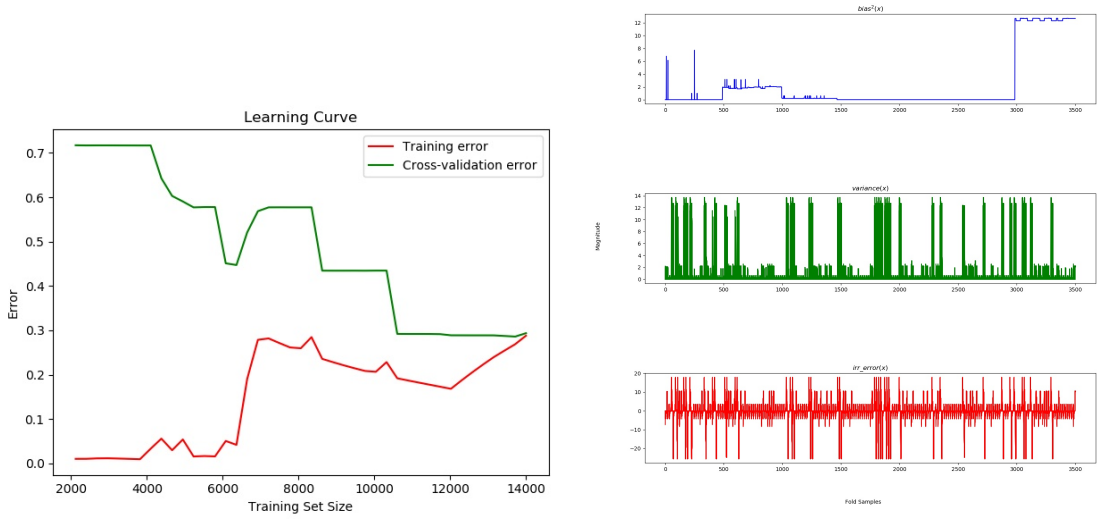
(a) ROC curve for SVM (b) Precision-Recall Curve for SVM

Fig. 4.9 ROC and P-R Curves for SVM

threshold value. Therefore, increasing the number of samples will not improve the model's performance. Hence, this model is not suited to detect forensic memory acquisition.

And the bias-variance decomposition plots in figure 4.10b show bias and variance are above threshold values. This is the reason why SVM classifier has higher number of FPs and FNs, and lower precision. The detection error rate (Δ_r) is calculated as the sum of bias and variance. For AdaBoost, Δ_r is 40 percent which is high. Therefore, SVM is not good at detecting forensic memory acquisition using memory parameters.

The rest of the ML classifiers Decision Tree, Gradient Boost, K-Neighbours and Random Forest perform well as they have ideal CM, high precision and recall, and optimal bias and variance as shown in table 4.2. Forensically, what this means is that, during the acquisition phase, for example, when the investigator is acquiring memory from a machine using the forensic tool FTK Imager, a high performing ML classifier such as Random Forest could detect memory acquisition with higher accuracy. And when memory acquisition is detected, then the attacker could proceed on to the next phase to defeat the acquisition process by shutting the machine down programatically as demonstrated in section 2.4.



(a) Learning Curve for SVM (b) Bias-Variance Decomposition for AdaBoost

Fig. 4.10 Learning and B-V Curves for AdaBoost

ML Classifier	Accuracy	Precision	Recall	F-Measure	Detection Error Rate
ADA Boost	84.28	31.48	45.65	37.26	73.4
Decision Tree	99.99	99.99	99.99	99.99	<5
Gaussian NB	96.74	89.31	88.4	88.85	18.4
Gradient Boost	99.99	99.98	99.98	99.98	<5
KNeighbours	99.99	99.97	99.97	99.97	<5
LDA	97.4	91.25	90.85	91.04	17.8
Random Forest	99.99	99.98	99.98	99.98	<5
SVM	91.68	62.88	49.78	55.56	40.0

Table 4.2 ML classifier comparison for Memory Features

4.3 Detecting Memory Acquisition using I/O Parameters

As for I/O parameters dataset, the size of the dataset is (17500,7), with 10,000 row iterations for the NAM category and 7500 iterations for the the AM category. Only part of the dataset with first 12 rows consisting of the FTK-NAM category is shown in figure 4.11.

other_bytes	other_count	read_bytes	read_count	write_bytes	write_count	class
136	49	27057	1	0	0	FTK-NAM
6534	724	136604	70	0	0	FTK-NAM
6534	724	136604	70	0	0	FTK-NAM
6534	724	136604	70	0	0	FTK-NAM
6534	724	136604	70	0	0	FTK-NAM
6534	724	136604	70	0	0	FTK-NAM
6534	724	136604	70	0	0	FTK-NAM
6534	724	136604	70	0	0	FTK-NAM
6534	724	136604	70	0	0	FTK-NAM
6534	724	136604	70	0	0	FTK-NAM
6534	724	136604	70	0	0	FTK-NAM

Fig. 4.11 IO Dataset

The complete I/O dataset is made available at bitbucket repository (Bitbucket, 2020b) for this research. There are 6 features in total for I/O parameters which are described in appendix C.1. The performance of SVM is poor when compared to RF classifier. First the confusion matrix will be analyzed to see what classes are affected the most, then ROC and P-R curves will be compared against each other to further support the results from the confusion matrix. Then by analysing learning curves and bias-variance decomposition curves the detection error rate is computed which further sheds light on the classifier's performance. As for SVM, the confusion matrix (SVM, 2020b) as shown in figure 4.12 during the first fold is considered, it can be seen that the classes 1, 3, 5 and 6 and majorily affected. This is due the fact that their either have higher number of FPs or FNs. For example, even though class 1 has correctly classified 472 instances whilst misclassifying 1544 of its instances which are FPs. It has misclassified around 13 instances of class 0, 10 instances of class 2, 492 instances of class 3, 502 and 532 instances of class 5 and class 6 respectively. Thereby, a total of 1544 misclassification instances. Which reflects that SVM does not do well detecting class 1. And for classes 3, 5 and 6 they have zero FPs. Now, if the ROC (SVM, 2020d) and P-R curves

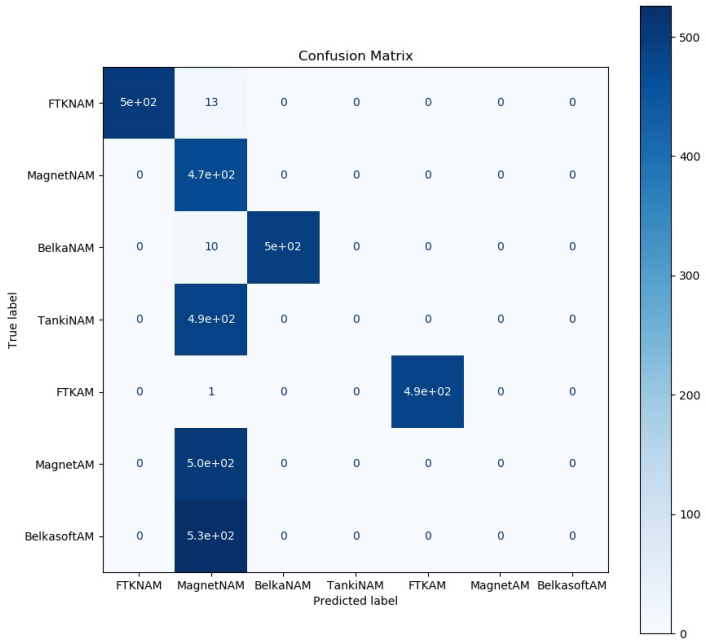
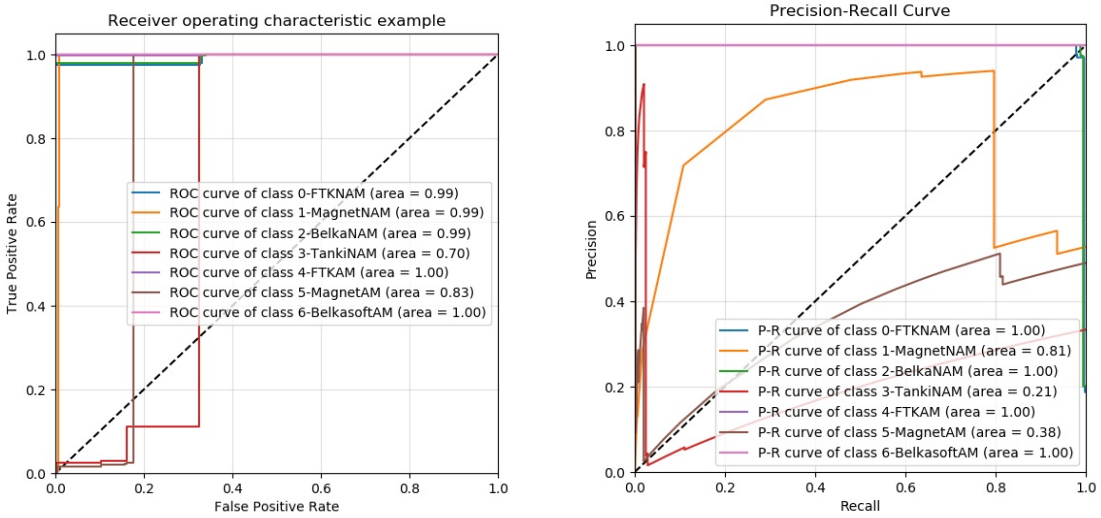
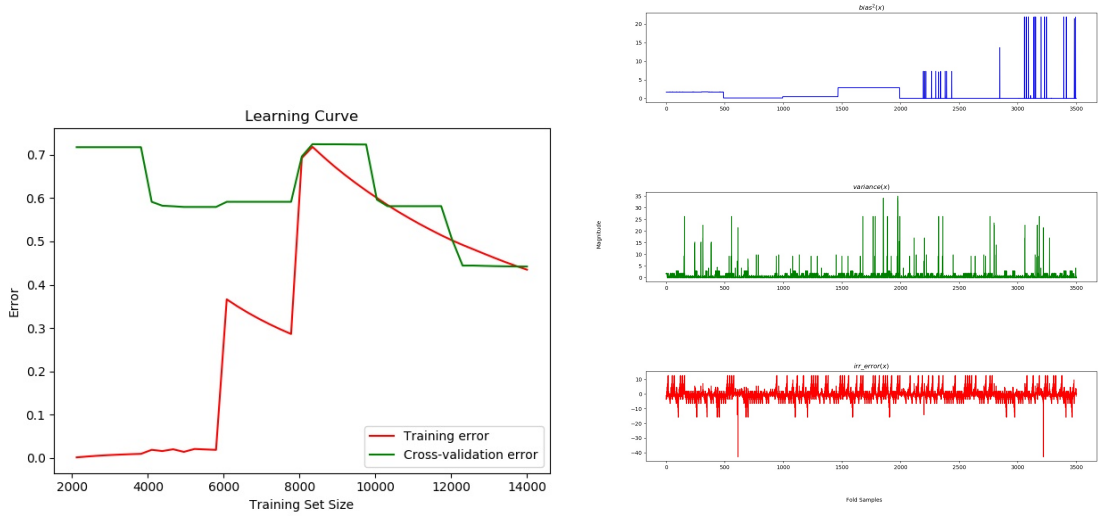


Fig. 4.12 Confusion Matrix for SVM Classifier



(a) ROC curve for SVM (b) Precision-Recall Curve for SVM

Fig. 4.13 ROC and P-R Curves for SVM



(a) Learning Curve for SVM (b) Bias-Variance Decomposition for SVM

Fig. 4.14 Learning and B-V Curves for SVM

(SVM, 2020c) for the first fold are compared, it is seen that the ROC curve shows that only classes 3 and 5 are affected which contradicts the results of the confusion matrix. Whereas, the P-R curve shows classes 1, 3, and 5 are affected but not class 6. Therefore, the results from the confusion matrix will be relied upon for only this case.

The learning curve (SVM, 2020a) shows that this model has high variance about 8000 training samples, and overall has a high bias as it is well above the threshold level. Also, high spikes at various points can be seen in bias-variance tradeoff graph of SVM as shown in figure 4.14 which indicates and further supports the results of the learning curve that this model does indeed suffer from high variance and high bias. Therefore, this model will not perform well at detecting memory acquisition when I/O parameters are used.

AdaBoost

As for AdaBoost, the confusion matrix (Ada, 2020d) as shown in figure 4.15, it can be seen that the classes, 1, 3, 5, and 6 are majorly affected. This is due the fact that their either have higher number of FPs or FNs. For example, even though class 2 has correctly

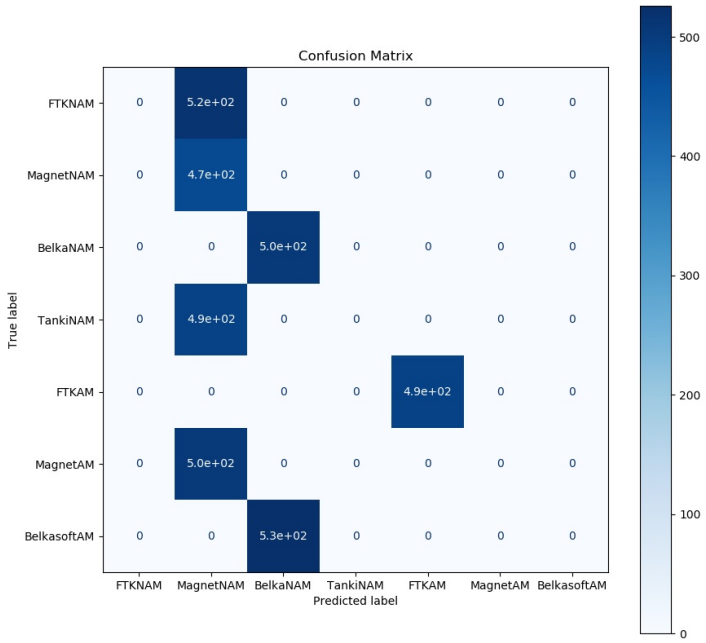
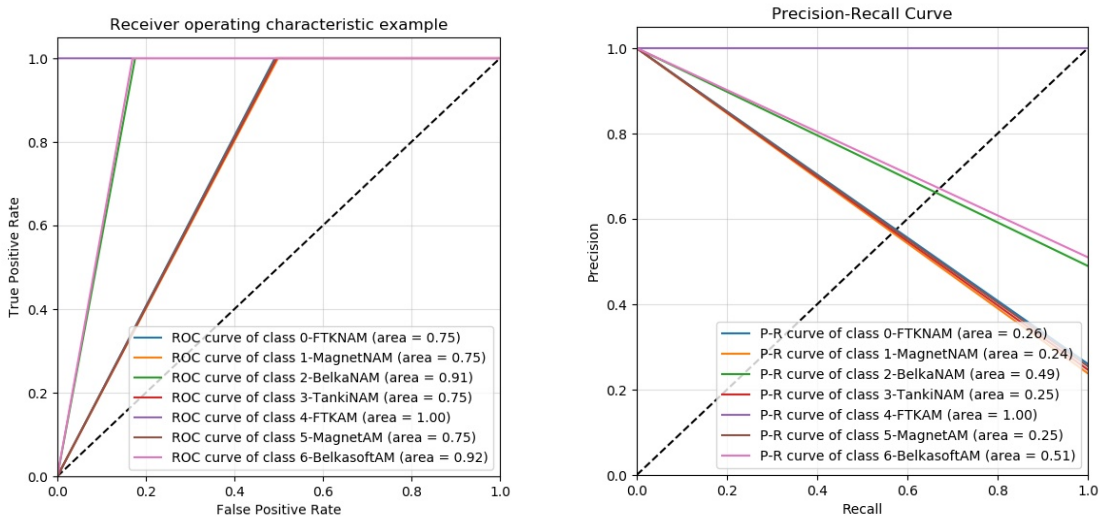


Fig. 4.15 Confusion Matrix for AdaBoost Classifier

classified 472 instances whilst misclassifying 1510 of its instances which are FPs. It has misclassfied around 520 instances of class 0, 490 instances of class 3, and 500 instances of class 5. Thereby, a total of 1510 misclassification instances which reflects that AdaBoost does not do well detecting class 1 and classes 3, 5 and 6 as they have zero FPs.

Now, if the ROC (Ada, 2020h) and P-R curves (Ada, 2020f) in 4.16 for the first fold are compared, it is seen that the ROC curve shows that 0, 1, 3, 5, and 6 is effected. This validates the results of the confusion matrix. As seen from the result of confusion matrix, that class 1 was the most effected class with 1510 FPs, this result is again supported by ROC curve in figure 4.16a and the P-R curve in 4.16b which has an area of just 0.26, which indicates that class 1 did not perform well.

The learning curve shows that this model has high variance until 9000 training samples, and overall has a high bias as it is well above the threshold level. Also, high spikes at



(a) ROC curve for AdaBoost (b) Precision-Recall Curve for AdaBoost

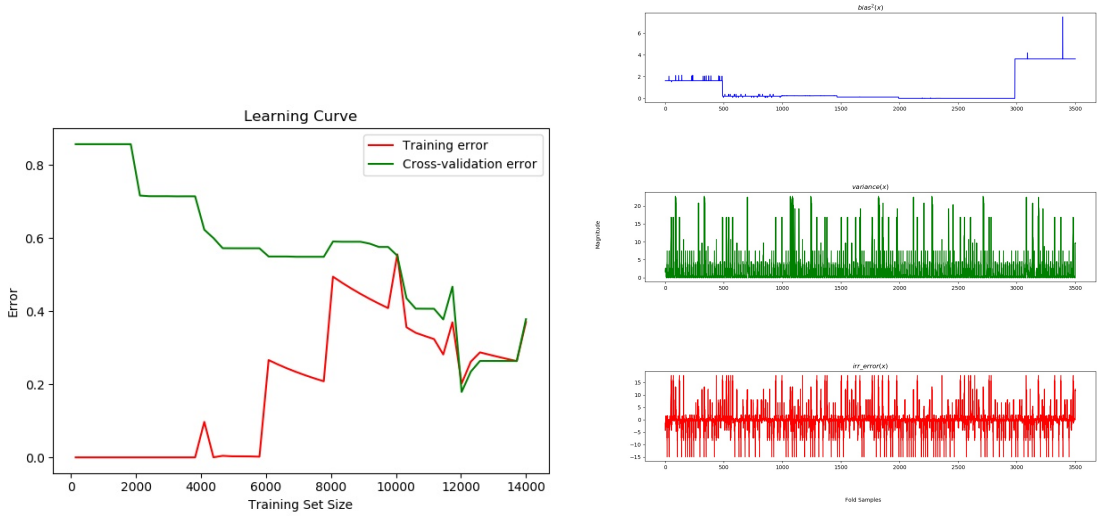
Fig. 4.16 ROC and P-R Curves for AdaBoost

various points can be seen in bias-variance tradeoff graph of AdaBoost as shown in figure 4.17 which indicates and further supports the results of the learning curve that this model does indeed suffer from high variance and high bias with a detection error rate (Δ_r) of 23.8 percent. Therefore, this model will not perform well at detecting memory acquisition when I/O parameters are used.

GaussianNB

As for GaussianNB, the confusion matrix as shown in figure 4.18, it can be seen that the classes 1, 3, and 5 and affected. This is due the fact that their either have higher number of FPs or FNs. For example, even though class 2 has correctly classified 472 instances whilst misclassifying 569 of its instances which are FPs. It has misclassified around 69 instances of class 3, and 500 instances of class 5. Which reflects that GaussianNB does not do well detecting class 1 and, classes 3 and 5 as they have zero FPs.

Now, if the ROC and P-R curves in 4.19 for the first fold are compared, it is seen that the ROC curve shows that 0, 1, 3, and 5 are effected. This validates the results of the confusion



(a) Learning Curve for AdaBoost (b) Bias-Variance Decomposition for AdaBoost

Fig. 4.17 Learning and B-V Curves for AdaBoost

matrix. As seen from the result of confusion matrix, that class 1 was the most effected class with 569 FPs, this result is again supported by ROC curve in figure 4.19a and the P-R curve in 4.19b which has an area of just 0.79, which is the lowest when compared to other classes.

The learning curve (Ada, 2020b) shows that this model has high variance upto 8000 training samples, and overall has a bias as it is above the threshold level. Also, high spikes at various points can be seen in bias-variance tradeoff graph of GaussianNB as shown in figure 4.20 which indicates and further supports the results of the learning curve that this model does indeed suffer from variance and bias with a detection error rate (Δ_r) of 19 percent. Therefore, this model will not perform well at detecting memory acquisition when I/O parameters are used.

LDA

As for LDA, the confusion matrix (LDA, 2020d)as shown in figure 4.21, it can be seen that the classes 1, 3, and 5 and affected. This is due the fact that their either have higher number of FPs or FNs. For example, even though class 1 has correctly classified 472 instances whilst

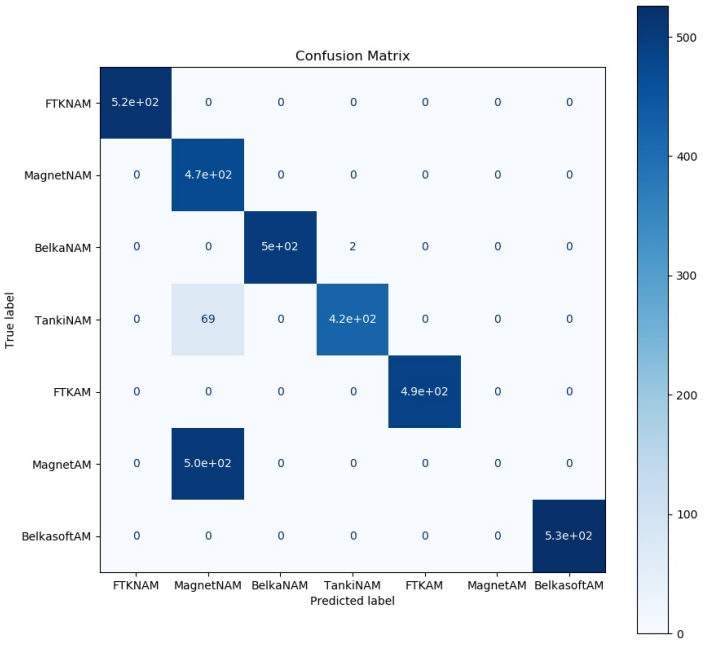
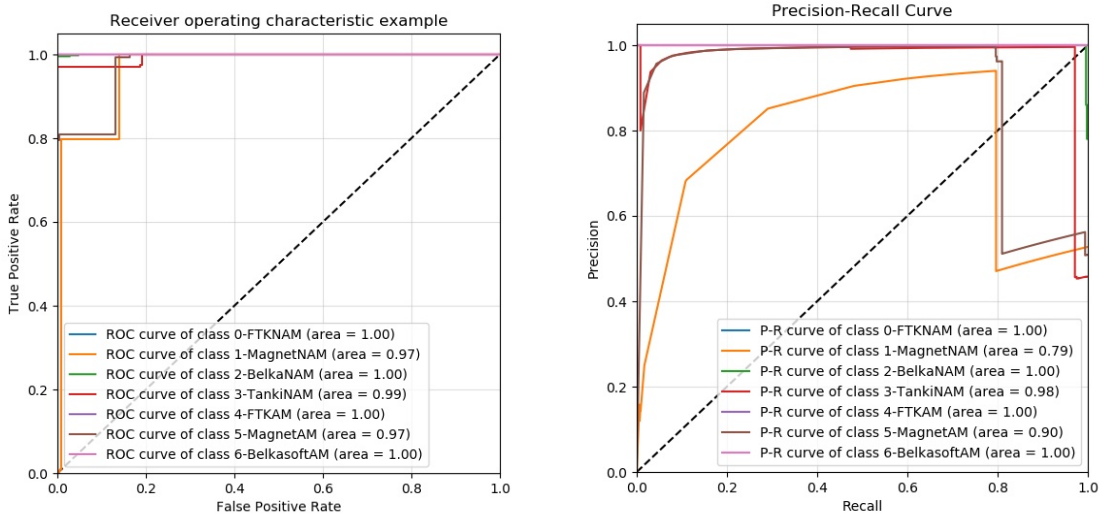
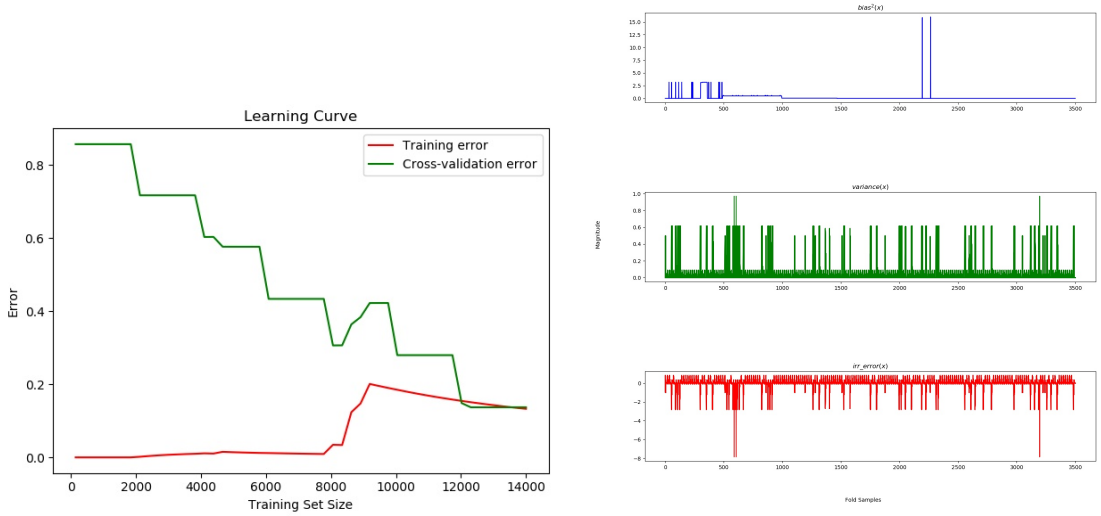


Fig. 4.18 Confusion Matrix for GaussianNB Classifier



(a) ROC curve for GaussianNB (b) Precision-Recall Curve for GaussianNB

Fig. 4.19 ROC and P-R Curves for GaussianNB



(a) Learning Curve for GaussianNB (b) Bias-Variance Decomposition for GaussianNB

Fig. 4.20 Learning and B-V Curves for GaussianNB

misclassifying 73 of its instances which are FPs. That is, it has misclassified 73 instances of class 1. Whereas, class 3 whilst having classified 430 of its instances, it has misclassified 32 instances of class 2 and 490 instances of class 5. This leads to class 5 having zero FPs. Which reflects that LDA does not do well detecting class 1 and, classes 3 and 5 as they have zero FPs.

Now, if the ROC (LDA, 2020h) and P-R curves (LDA, 2020f) in 4.22 for the first fold are compared, it is seen that the ROC curve shows that classes 1, 3, and 5 are effected. This validates the results of the confusion matrix. As seen from the result of confusion matrix, that class 3 was the most effected class with 490 FPs, this result is again supported by ROC curve in figure 4.22a and the P-R curve in figure 4.22b which has an area of just 0.35, which is the lowest when compared to other classes.

The learning curve (LDA, 2020b) shows that this model has high variance upto 12000 training samples, and overall has a bias as it is above the threshold level. Also, high spikes at various points can be seen in bias-variance tradeoff graph of LDA as shown in figure 4.23 which indicates and further supports the results of the learning curve that this model does

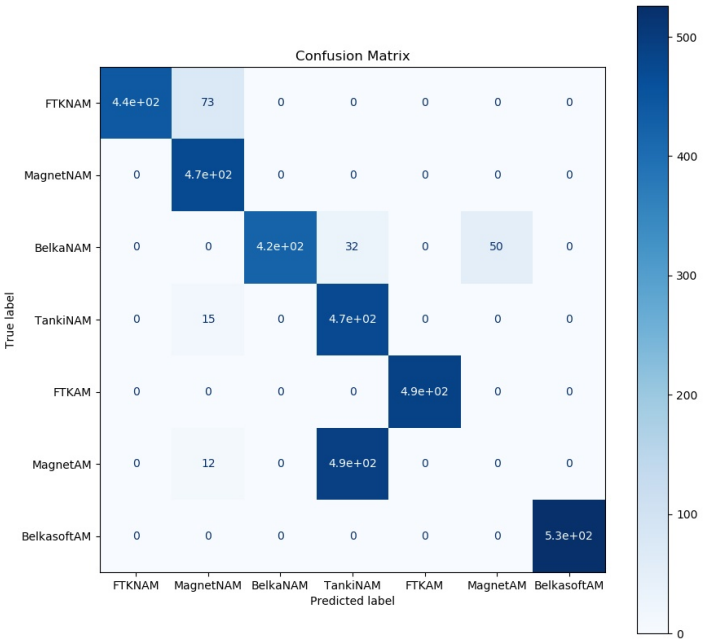
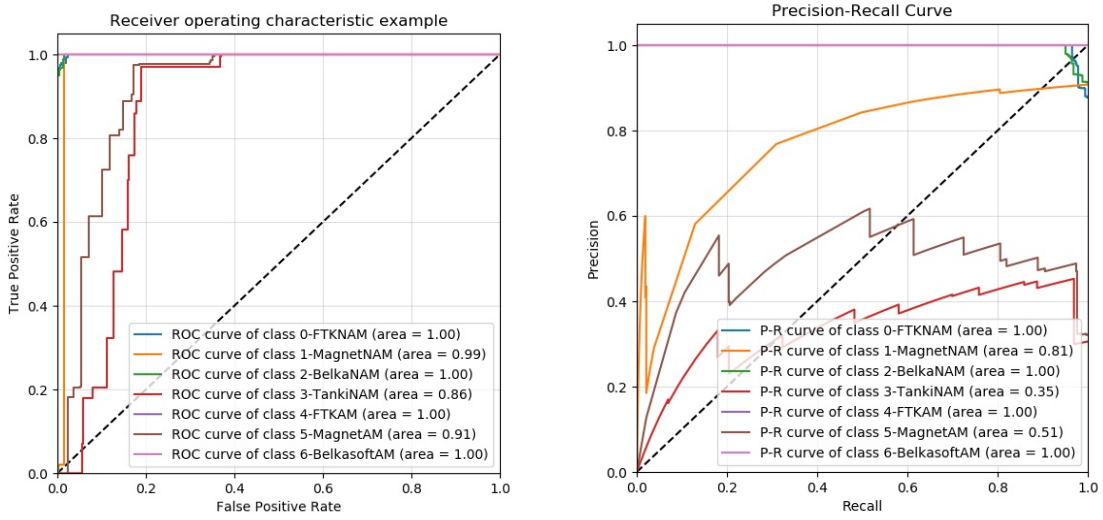
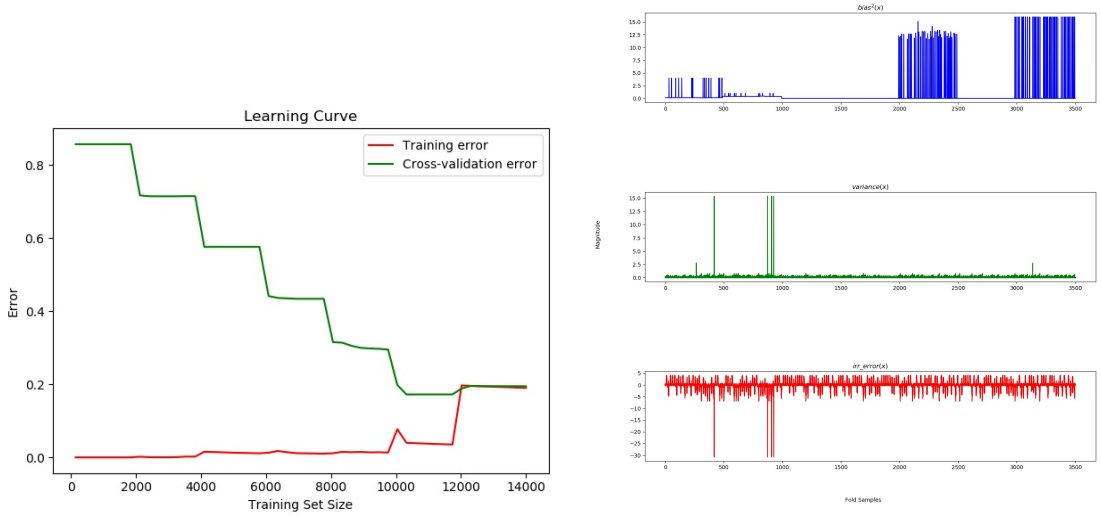


Fig. 4.21 Confusion Matrix for LDA Classifier



(a) ROC curve for LDA (b) Precision-Recall Curve for LDA

Fig. 4.22 ROC and P-R Curves for LDA



(a) Learning Curve for LDA (b) Bias-Variance Decomposition for LDA

Fig. 4.23 Learning and B-V Curves for LDA

indeed suffer from variance and bias with a detection error rate (Δ_r) of 78 percent. Therefore, this model will not perform well at detecting memory acquisition when I/O parameters are used. The rest of the ML classifiers Decision Tree, Gradient Boost, K-Neighbours and RandomForest perform well as they have ideal CM, high precision and recall, and optimal bias and variance as shown in table 4.3. When compared to memory parameters, the I/O parameters perform well, and this has an impact on what parameters to use to defeat the forensic acquisition process. For example, to accqurately detect forensic memory acquisiton with higher precision, recall and lower detection rate, an attacker would choose to execute the detection using a ML classifier such as random forest. And after detection, the forensic memory acquisition process could be defeated by passively shutting the machine down as seen in section 2.4.

ML Classifier	Accuracy	Precision	Recall	F-Measure	Detection Error Rate
AdaBoost	89.25	51.48	62.82	56.58	23.8
Decision Tree	99.99	99.99	99.99	99.99	<5
GaussianNB	96.17	81.91	86.68	84.22	19
Gradient Boost	99.99	99.99	99.99	99.99	<5
KNeighbours	99	99	99	99	<5
LDA	94.37	75.62	80.88	78.16	78
Random Forest	99.99	99.99	99.99	99.99	<5
SVM	87.51	46.22	56.37	50.79	15.7

Table 4.3 Comparison of ML Classifiers based on I/O features

4.4 Detecting Memory Acquisition using CPU Parameters

In this section, only the analysis of top three least performing models are discussed. When compared to memory parameters and I/O parameters, all models do not perform well using CPU parameters. The reason for this could be fewer number of features in the CPU parameters (Appendix C.2). The size of the CPU dataset is (17500, 5) and the first 10000 row iterations belong to the NAM category and the next 7500 iterations belong to the AM category.

blocking	kernel	nonblocking	user	class
12.7	1.6875	0	0.203125	FTK-AM
12.5	2.703125	12.525	0.265625	FTK-AM
12.3	3.734375	12.4	0.28125	FTK-AM
12.7	4.65625	6.3	0.390625	FTK-AM
12.7	5.671875	12.6	0.421875	FTK-AM
12.5	6.6875	12.4	0.46875	FTK-AM
12.5	7.671875	12.6	0.515625	FTK-AM
13.475	8.625	12.2125	0.65625	FTK-AM
12.7	9.625	12.6	0.703125	FTK-AM
12.7	10.65625	12.6	0.71875	FTK-AM

Fig. 4.24 CPU Dataset

Only a part of the dataset is shown in figure. The complete CPU dataset is available on the bitbucket repository (Bitbucket, 2020a). Behaviour of the models are discussed next followed with the performance metrics which is shown in table 4.4.

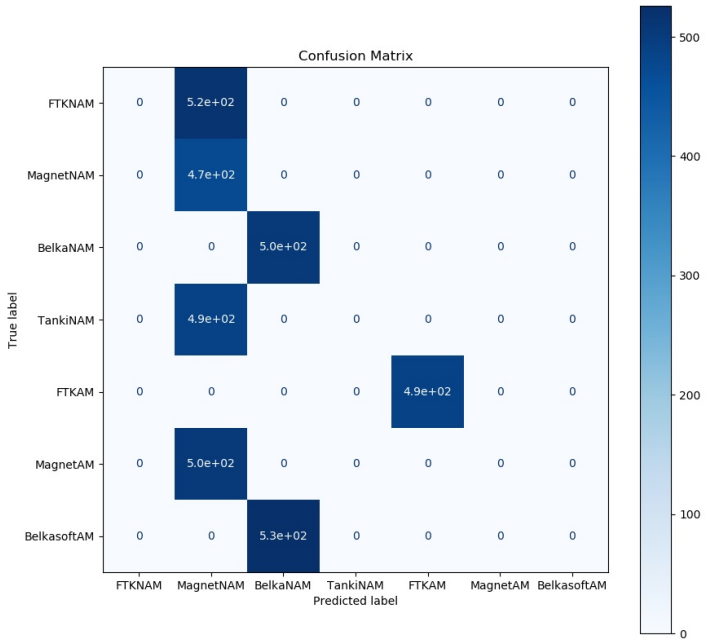
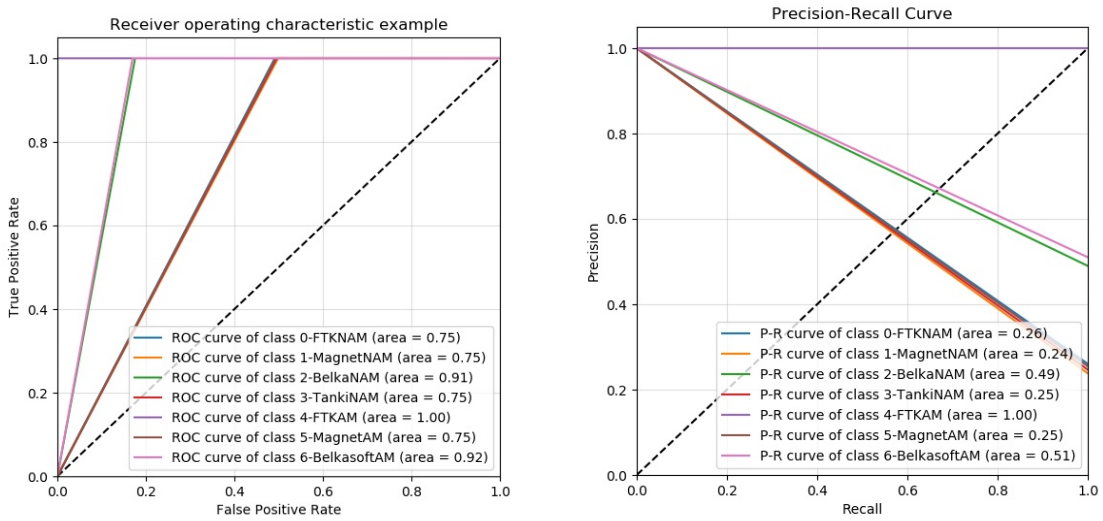


Fig. 4.25 Confusion Matrix for AdaBoost Classifier

Ada Boost

The Ada Boost model shows the least performance when compared to other models. When analysing the confusion matrix of Ada Boost classifier (Ada, 2020c)as shown in figure 4.25, it can be seen that for the first fold classes 0, 1, 2, 4, and 5 are highly affected. Class 0, 2, and 5 have 505 FNs, whilst class 1 has 505 FPs. Among all classes, class 4 is majorly effected with 982 FPs and 64 FNs. The results of the confusion matrix shows that this model's performance is poor. To further support this result, the ROC (Ada, 2020g) and P-R curves (Ada, 2020e) will be compared, and also the learning curve and bias-variance decomposition graphs will be analyzed to guage the type of bias and variance this model exhibits and the detection error rate respectively. When ROC and P-R curves are compared together as seen in figure 4.26, it is observed that classes 0, 1,2, 4, and 5 are effected. As seen during the



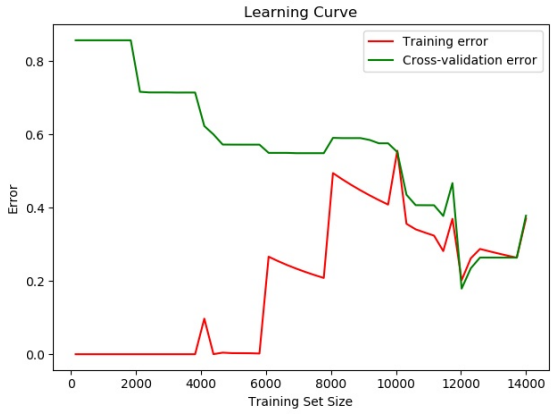
(a) ROC curve for AdaBoost (b) Precision-Recall Curve for AdaBoost

Fig. 4.26 ROC and P-R Curves for AdaBoost

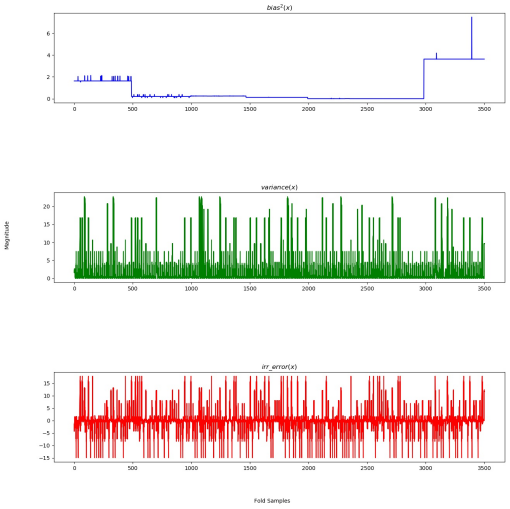
analysis of confusion matrix that class 4 has been highly effected due to higher number of FPs, this result is supported by ROC curve which shows that class 4 has the least area under the curve, and class 4 also has the least precision and recall with area under the P-R curve being 0.30 which is low. Therefore, the comparison of ROC and P-R curves further support the results from confusion matrix with regards to the classes effected dur to FPs and FNs.

The learning curve (Ada, 2020a) in figure 4.27a shows this model suffers with high variance and high bias. The model exhibits high variance upto 8000 samples and continues to have variance. It has high bias as both the curves meet at approximately at a 50 percent error value with a detection error rate (Δ_r) of 99 percent. Therefore, this model does not perform well using CPU parameters.

In table 4.4 the performance of all ML classifiers are shown. Also, in table 4.4 it can be seen that all ML classifiers do not perform well using CPU parameters.



(a) Learning curve for AdaBoost



(b) Bias-Variance Curve for AdaBoost

Fig. 4.27 LC and B-V Curves for AdaBoost

ML Classifier	Accuracy	Precision	Recall	F-Measure	Detection Error Rate
AdaBoost	60.58	46.11	53.51	49.53	79
Decision Tree	97.2	89.97	89.91	89.93	17
GaussianNB	95.34	84.45	83.65	84.04	36
Gradient Boost	97.42	91.08	91.17	91.12	13
K Neighbours	96.25	90.65	87.28	88.93	19
LDA	91.37	66.91	70.02	68.42	81
Random Forest	95.34	84.45	83.65	84.04	14
SVM	94.57	74.94	81.14	77.91	28

Table 4.4 Performance of ML Classifiers for CPU features

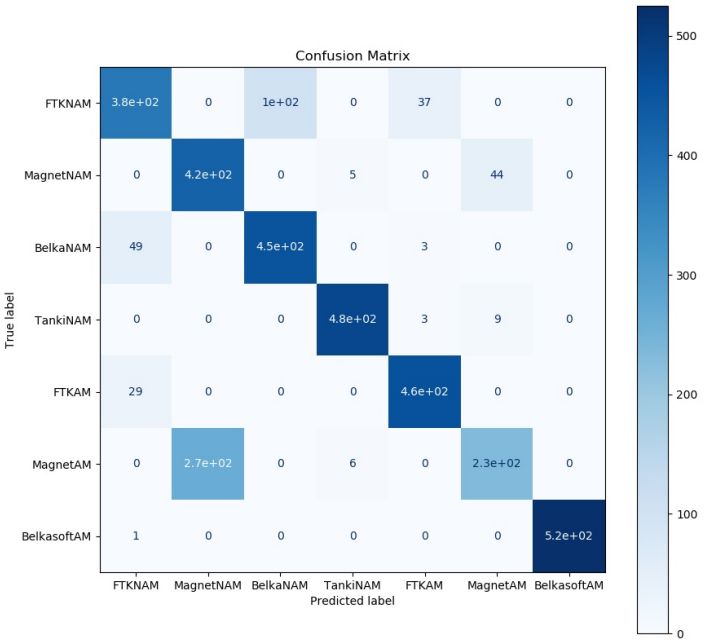
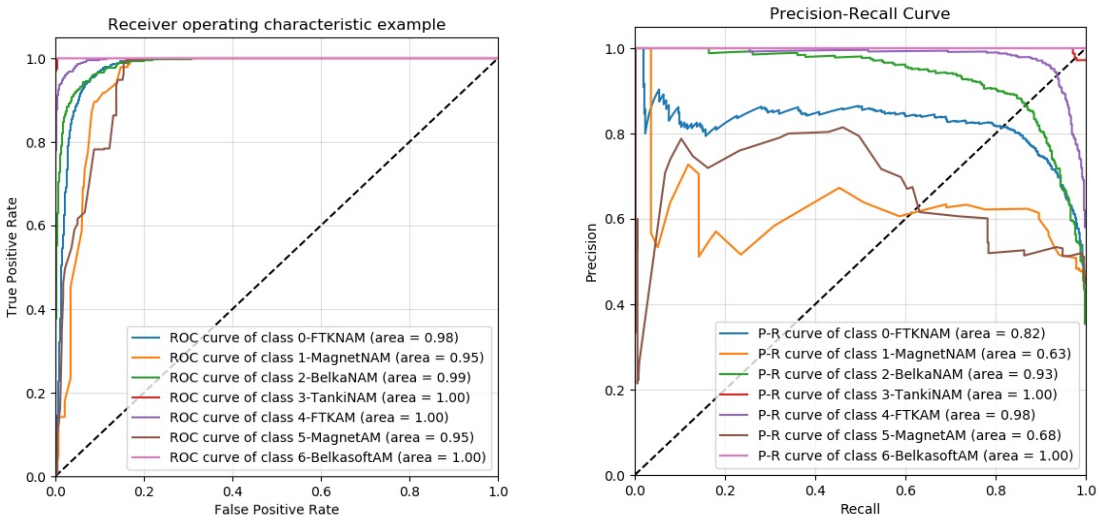


Fig. 4.28 Confusion Matrix for GaussianNB Classifier

GaussianNB

When analysing the confusion matrix of GaussianNB classifier (Gau, 2020b) as shown in figure 4.28, it can be seen that classes 0, 1, 2, 4, and 5 are affected. Class 0, 1 and 2 have 79, 270, and 100 FPs respectively. Whereas, classes 1 and 5 have 137 and 276 FNs respectively. The results of the confusion matrix shows that this model's performance is poor. To further support this result, the ROC and P-R curves will be compared, and also the learning curve and bias-variance decomposition graphs will be analyzed to guage the type of bias and variance this model exhibits and the detection error rate respectively. When ROC (Gau, 2020e)and P-R curves (Gau, 2020c) in figure 4.29 are compared together, it is observed that classes 0, 1, 4, and 5 are effected. As seen during the analysis of confusion matrix that class 1 and 5 have been effected due to higher number of FNs, this result is supported by ROC curve which



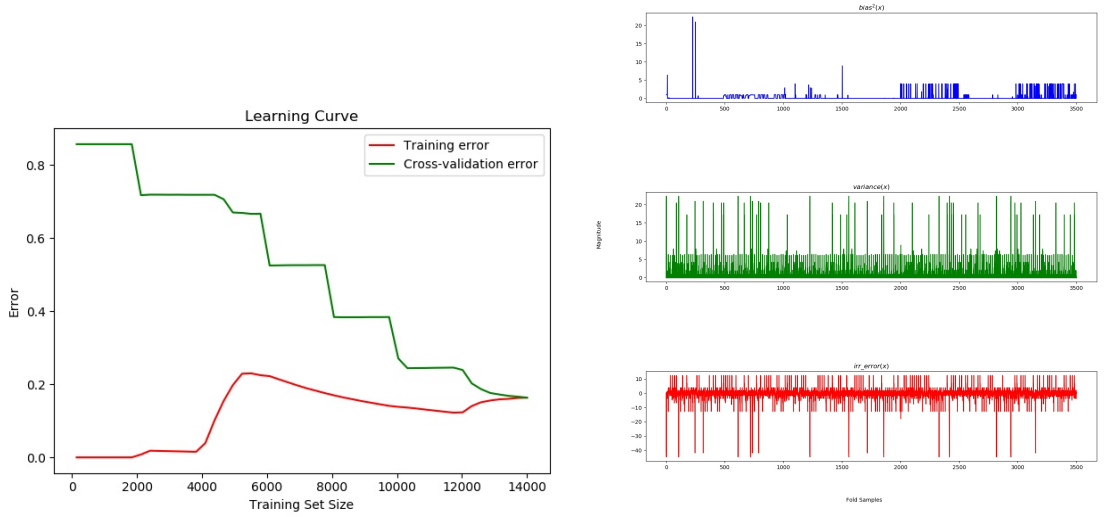
(a) ROC curve for GaussianNB (b) Precision-Recall Curve for GaussianNB

Fig. 4.29 ROC and P-R Curves for GaussianNB

shows that class 1 and 5 have the least area under the curve when compared to other classes, and class 1 and 5 also has the least precision and recall with area under the P-R curve being 0.63 and 0.68 respectively, which is low. Therefore, the comparison of ROC and P-R curves further support the results from confusion matrix with regards to the classes effected dur to FPs and FNs. The learning curve (Gau, 2020a) in figure 4.30a shows this model suffers with variance and bias. The model exhibits variance up to 12000 samples. It has high bias as both the curves meet at 20 percent error value with a detection rate of Δ_r of 36 percent. Therefore, this model does not perform well using CPU parameters.

LDA

When analysing the confusion matrix of LDA classifier (LDA, 2020c) as shown in figure 4.31, it can be seen that classes 0, 1, 2, 3, 4, and 5 are affected. Class 0, 1, 2 and 4 have 56, 760, 71, and 96 FPs respectively. Whereas, classes 0 and 5 have 130 and 500 FNs respectively. The results of the confusion matrix shows that this model's performance is poor. To further support this result, the ROC (LDA, 2020g) and P-R curves (LDA, 2020e) will



(a) Learning curve for GaussianNB

(b) Bias-Variance Curve for GaussianNB

Fig. 4.30 LC and B-V Curves for GaussianNB

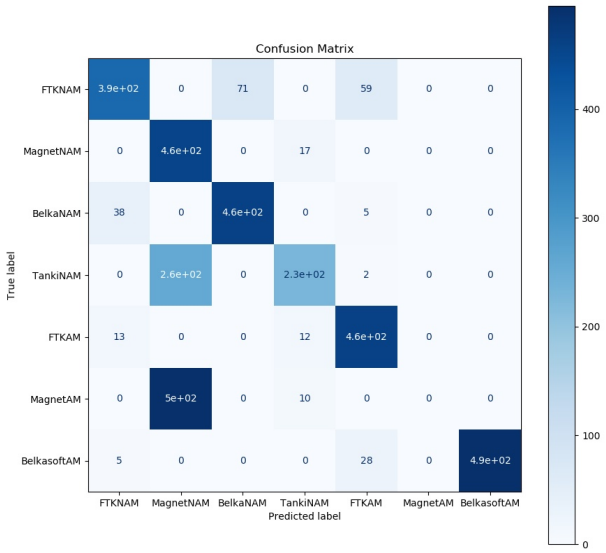
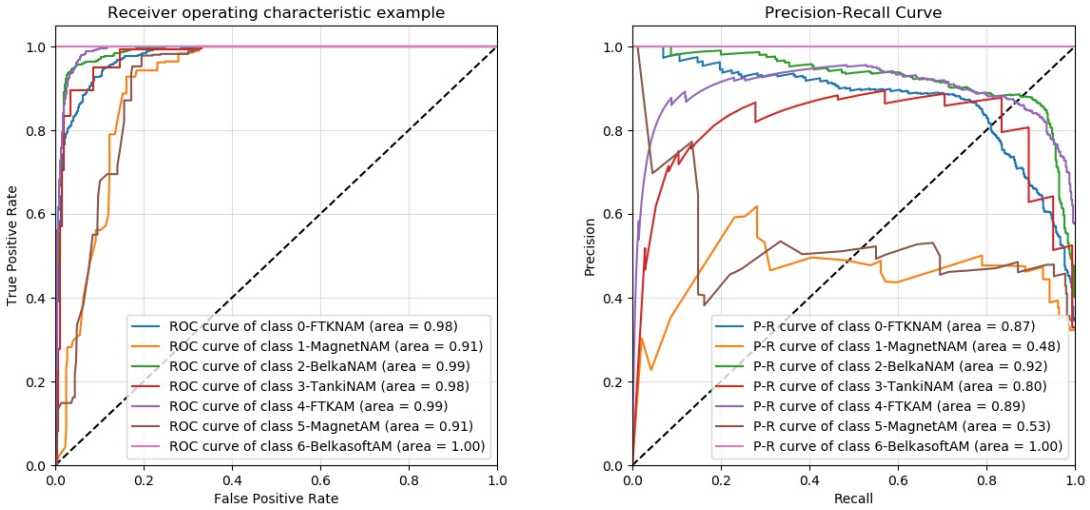


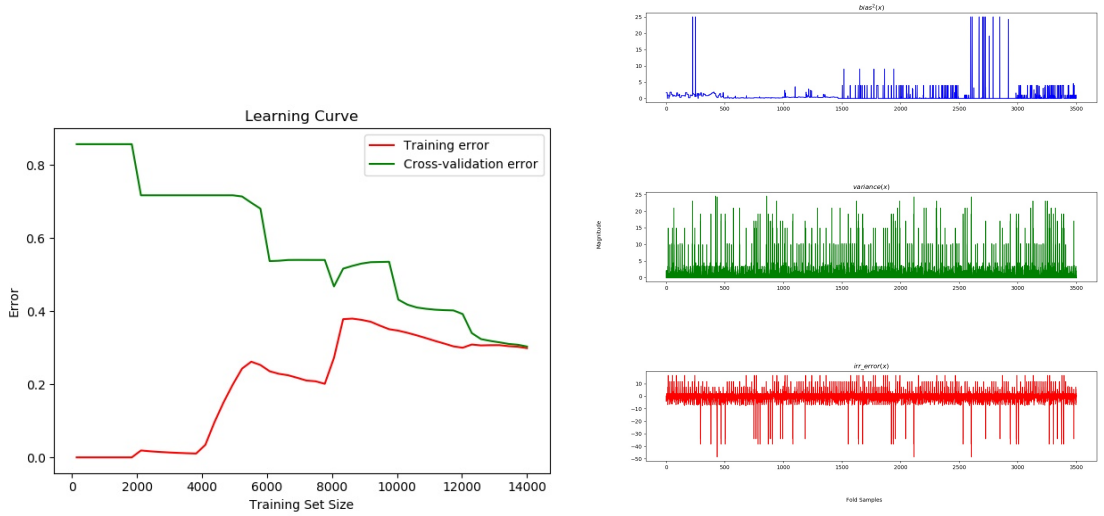
Fig. 4.31 Confusion Matrix for LDA Classifier



(a) ROC curve for LDA (b) Precision-Recall Curve for LDA

Fig. 4.32 ROC and P-R Curves for LDA

be compared along with learning curve and bias-variance decomposition which will also be analyzed to gauge the type of bias and variance this model exhibits and the detection error rate respectively. When ROC and P-R curves in figure 4.32 are compared together, it is observed that classes 0, 1, and 5 are effected. As seen during the analysis of confusion matrix that class 0 and 5 have been effected due to higher number of FNs, this result is supported by ROC curve which shows that class 1 and 5 have the least area under the curve when compared to other classes, and class 1 and 5 also has the least precision and recall with area under the P-R curve being 0.48 and 0.53 respectively, which is low. Therefore, the comparison of ROC and P-R curves further support the results from confusion matrix with regards to the classes effected due to FPs and FNs. The learning curve (LDA, 2020a) in figure 4.33a shows this model suffers with variance and bias. The model exhibits variance upto 12000 samples. It has high bias as both the curves meet at approximately at a 30 percent error value with a detection error rate Δ_r of 81 percent. Therefore, this model does not perform well using CPU parameters.



(a) Learning curve for LDA

(b) Bias-Variance Curve for LDA

Fig. 4.33 LC and B-V Curves for LDA

4.5 Integrated Analysis of MIOC Parameters

The memory, I/O, and CPU (MIOC) parameters were combined to create one single dataset, the MIOC dataset. The size of the dataset is (17500,21), it contains 20 features which includes 10, 6, and 4 from memory, I/O, and CPU features respectively. The dataset is too large to be included in this thesis. The MIOC dataset is available at the bitbucket repository (Bitbucket, 2020c). The reason for combining all the three parameters was to observe if there would be a decrease or increase in the performance of the ML classifiers. Table 4.5 shows the comparison of performance of ML classifiers for integrated MIOC parameters.From this table it can be seen that Ada Boost and SVM classifier have underperformed. Whereas, the rest of the ML classifiers perform well. The reason for poor performance for both Ada Boost and SVM classifiers is that they have low precision and recall as seen in figure 4.5. This is due to the fact that they exhibit high bias and variance (Appendix A.1).

When compared to memory, I/O, and CPU parameters individually, the intergrated MIOC parameters perform well. Table 4.6 shows the percentage increase of integrated MIOC

ML Classifiers	Accuracy	Precision	Recall	F-Measure	Detection Error Rate
Ada Boost	88.45	48.37	60	53.56	60.4
Decision Tree	99.99	99.99	99.99	99.99	<5
GaussianNB	99.98	99.42	99.48	99.44	<5
Gradient Boost	99.99	99.99	99.99	99.99	<5
KNeighbours	99.99	99.85	99.57	99.70	<5
LDA	99.98	99.4	99.37	99.38	<5
Random Forest	99.99	99.99	99.99	99.99	<5
SVM	91.62	61.68	70.65	65.86	71

Table 4.5 Integrated MIOC parameters performance

ML Classifiers	Accuracy	Precision	Recall	F-Measure	Detection Error Rate
Ada Boost	4.17	16.89	14.35	15.51	-13
Decision Tree	0	0	0	0	0
GaussianNB	3.24	10.11	11.08	10.57	-17.9
Gradient Boost	0	0.01	0.01	0.01	0
KNeighbours	0	-0.12	-0.4	-0.18	0
LDA	2.58	8.15	8.52	8.33	-17.3
Random Forest	0	0.01	0.01	0.01	0
SVM	-0.06	-1.2	20.87	-2.269	-32.9

Table 4.6 Percentage Increase over Memory Parameters

parameters performance when compared to memory parameters. Ada Boost classifier shows an 4.17 percent increase in accuracy and, 16.89 and 14.35 percent increase in precision and recall respectively. Whereas, the detection error rate (Δ_R) shows an 13 percent decrease, which means this classifier detection performance is better when MIOC parameters are combined compared to detecting memory acquisition individually with memory parameters. Interestingly, SVM classifier shows a slight decrease in accuracy and precision, but this is compensated by 20.87 increase in recall and 32.9 percent decrease in detection error rate. Similarly, other classifiers also show an increase in performance when MIOC parameters are combined. The only exception to this case is the KNeighbours classifier, which shows a marginal decrease in precision and recall.

When compared to the I/O parameters in table 4.7, Ada Boost classifier's detection error rate decreases by 36.6 percent with slight increase in precision, recall, and accuracy which

ML Classifiers	Accuracy	Precision	Recall	F-Measure	Detection Error Rate
Ada Boost	0.8	3.11	2.82	2.95	-36.6
Decision Tree	0	0	0	0	0
GaussianNB	-3.81	-17.51	-12.8	-14.78	14
Gradient Boost	0	0	0	0	0
KNeighbours	-0.99	-0.85	-0.57	-0.68	0
LDA	-5.61	-23.78	-18.49	20.80	73
Random Forest	0	0	0	0	0
SVM	-4.11	-15.46	-14.28	-14.84	-55.3

Table 4.7 Percentage Increase over I/O Parameters

are 3.11, 2.82, and 0.8 percent respectively. Whereas, Gaussian NB and LDA show a major decrease in performance. The accuracy, precision, and recall of GaussianNB decrease, which increases the detection error rate by 14 percent. Similarly, for LDA the detection error rate drastically increases by 73 percent with an decrease in accuracy, precision, and recall. The performance of SVM classifier shows a strange result because as the accuracy, precision, and recall increase, the detection error rate is expected to increase, but it decreases by 55.3 percent. The integrated parameters performance shows a significant increase in performance over CPU parameters for all ML classifiers, but the only exception being SVM classifier which shows a decrease in performance (table 4.8). The recall and precision decrease by 13.26 percent and 10.49 percent respectively which increases the detection error rate by 43 percent. When compared to all other ML classifiers except for SVM, they all show an increase in accuracy, precision and recall, and a decrease in detection error rate. LDA classifier shows a considerable decrease in detection error rate which is 76 percent followed by Ada Boost with an decrease of 38.6 percent.

4.6 Conclusion

The analysis of the individual MIOC parameters results show that ADA boost and SVM classifiers are not good at detecting forensic memory acquisition as various classes show

ML Classifier	Accuracy	Precision	Recall	F-Measure	Detection Error Rate
AdaBoost	27.87	2.26	6.49	3.35	-38.6
Decision Tree	2.79	10.02	10.08	10.04	-12
GaussianNB	4.64	14.97	15.83	15.38	-31
Gradient Boost	2.57	8.91	8.82	8.86	-8
K Neighbours	3.74	9.2	12.29	10.52	-14
LDA	8.61	32.49	29.35	30.84	-76
Random Forest	4.65	15.54	16.34	15.92	-9
SVM	-2.95	-13.26	-10.49	-11.71	43

Table 4.8 Percentage increase over CPU Parameters

either higher number of false positive or false negative with lower precision and recall rates. The reason for this higher FPs and FNs is that these models either have higher bias or variance. Therefore, these models are not recommended to detect forensic memory acquisition. Whereas, the random forest classifier has fewer number of false positives and false negatives across all folds, and also high precision and recall. This is because the classifier exhibits low bias and variance, and hence Random Forest Classifier is recommended to detect forensic memory acquisition.

When compared to the integrated MIOC parameters, performance of individual MIOC parameters show an increase in accuracy, precision and recall, and a decrease in the detection error rate. This signifies the relevance of integrated MIOC parameters in addressing the performance issues in individual MIOC parameters. In relation to digital forensic investigation process, detecting forensic memory acquisition using integrated MIOC parameters for example, could defeat the acquisition process by shutting down the machine. In which case, the examiner will not have evidence pertaining to memory to work with. Also, instead of shutting the machine down, if memory acquisition is detected, the evidence could be tampered with, and this could make the investigator deduce false conclusions. Therefore, defeating the forensic process of acquisition and analysis.

Chapter 5

Detecting Forensic Memory Acquisition Patterns using Convolution Neural Network

5.1 Introduction

In Chapter 4, the results support the hypothesis mentioned in 3.3.2. This chapter proposes another way to detect forensic memory acquisition to further support the results from Chapter 4. The approach used in this study is that of (Wang and Oates, 2015), who first proposed to convert timeseries data to GAF images and then used tiled CNN to learn features from their dataset. Their approach produced competitive results for a small parameter space. Therefore, the MIOC data set is transformed into images and then a CNN model is used to identify the DNA patterns in the datasets. The results show that the model detects DNA patterns from the I/O data with a greater accuracy, precision, and recall when compared to memory and CPU dataset respectively.

5.2 Distinctive Native Attribute Patterns

In this section, a method to transform features to distinctive native attribute (DNA) patterns is presented. Features can be transformed to DNA patterns by encoding time series as images.

In this thesis, the DNA patterns are generated by making use of gramian summation field (GSF) aspect of the gramian angular field (GAF) (Wang and Oates, 2015). GAF can be used to encode multivariate timeseries to DNA patterns as demonstrated in figure 5.1. In a GAF conversion, first the timeseries in figure 5.1a is transformed to polar co-ordinate system as shown in figure 5.1b, and then finally to GSF (Figure 5.1c). This process is explained in detail in subsection 5.2.1.

5.2.1 Gramian Angular Field

Let the timeseries data be represented as $X = \{x_1, x_2, \ldots, x_n\}$ where all the data elements of X are real valued observations. And, the rescale equation of X between the interval $[-1, 1]$ is:

$$\bar{x_i} = \frac{(x_i - max(X)) + (x_i - min(X))}{max(X) - min(X)} \tag{5.1}$$

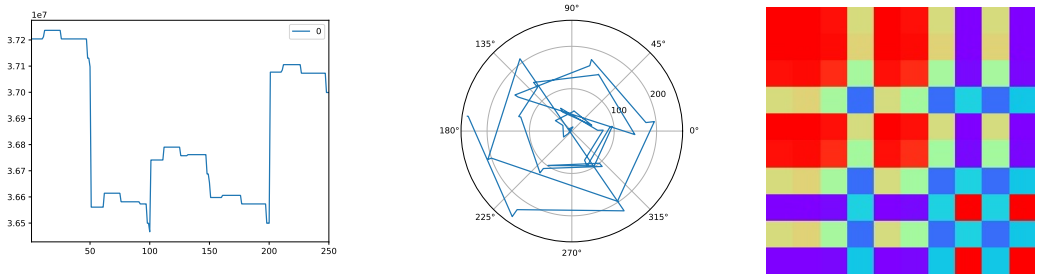
Now, to represent the rescaled timeseries $\bar{X}$ in polar co-ordinate system, the values must be encoded by angular cosine (ϕ) and time-intervals as the radius as shown in equation 5.2:

$$\begin{aligned} \phi = arccos(\bar{x_i}), -1 \leq \bar{x_i} \leq, \bar{x_i} \in \bar{X} \\ r = \frac{t_i}{N}, t_i \in N \end{aligned} \tag{5.2}$$

In equation 5.2, t_i is the time interval between data values, and N is the constant that is used to regularize the span of polar co-ordinate system. The advantage of equation 5.2 is that when compared to the cartesian co-ordinate system, polar co-ordinates preserve temporal relations. Now, the angular properties of the rescaled timeseries can be utilized by taking the trigonometric sum between each point within time intervals to identify temporal correlation to generate GAF. The GAF is represented as:

$$G = \begin{bmatrix} cos(\phi_{1,1}) & cos(\phi_{1,2}) & \dots & cos(\phi_{1,n}) \\ cos(\phi_{2,1}) & cos(\phi_{2,2}) & \dots & cos(\phi_{2,n}) \\ \vdots & \ddots & \vdots & \vdots \\ cos(\phi_{n,1}) & cos(\phi_{n,2}) & \dots & cos(\phi_{n,n}) \end{bmatrix} \tag{5.3}$$

Each element in the gram matrix G can be represented by the inner dot product as $x.y - \sqrt{(}1-x^2).(\sqrt{(}1-y^2)$, which is the definition of GSF. The GSF mainly has an essential property, and is reason why it is used in this research. The property of GSF is that, it preserves the temporal dependency between MIOC parameters and forensic memory acquisition. This is because GSF is bijective when $0 \leq \phi \leq \pi$, which means GSF can be transformed back to the normalized time series (Tsai et al., 2019). It is this temporal dependency property of GSF this study takes advantage of to support the prediction made in 3.3.3.



(a) Timeseries (b) Polar Chart (c) GSF

Fig. 5.1 DNA pattern from timeseries

A timeseries can be reconstructed approximately with the help of the main diagonal element $G_{i,i}$ when $k = 0$ from the high level features learned by the deep neural network. The only disadvantage is that the dimensions of GAF increases as the size of the gramian matrix is $n \times n$ when the length of the raw time series is n. To address this issue the timeseries is smoothed with the application of piecewise aggregation approximation whilst preserving its trends (Wang and Oates, 2015). The process of generating DNA pattern from timeseries is shown in figure 5.1.

Once the DNA patterns have been generated, it can be analyzed using convolution neural network to automatically extract features to distinguish between forensic memory acquisition or non-acquisition. In which case the binary classification strategy is adopted. If a forensic software tool has to be classified when memory acquisition takes place then the multiclassification strategy will be used.

5.3 Convolution Neural Network

CNN is a deep neural network mostly applied to analyze visual images (Tsai et al., 2019), and it employs a specialized linear operation known as convolution in its neural network. CNN consists of an input layer, multiple hidden layers, and an output layer. These layers are described in the following subsections.

5.3.1 Input Layer

The input layer takes a three-dimensional (3D) input of the form (rows $\times$ cols $\times$ channels). If the input layer is receiving a colour image, then the depth or channel would be 3, because the 3 channels would contain red, green, and blue colours as shown in figure 5.2. Each channel in turn consists of 4 rows and 4 columns; therefore, the input size would be ($4\times4\times3$). If the number of channels is 1 in case of grayscale images, in which case the input size would be ($4\times4\times1$). After the input size is determined, the image is processed by the hidden layer.

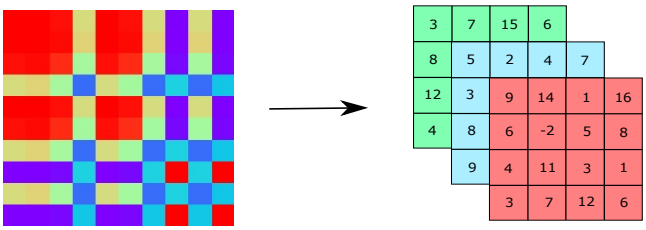


Fig. 5.2 Three-Channel (R,G,B) input layer

5.3.2 Hidden Layers

The hidden layers which consist of convolution layers, pooling layers and fully connected layers. These layers are described below:

Convolution Layer

The convolution layer uses the mathematical principle of convolution. Since, the image is a two-dimensional (2D) signal, 2D convolution will be applied. The 2D convolution with image (I) and kernel (K)is expressed by equation 5.4 (Goodfellow et al., 2016, Pg 328):

$$(I*K)(i,j) = \sum_{m,n} K(m,n)I(i+n, j+m) \tag{5.4}$$

The 2D convolution drags a kernel or filter over the image. The dragging starts at the origin of the image and ends at the point where the pixels end for the image. The kernel shifts over the image by specific number of pixels called the stride 's'. At times, zero-padding 'p' is added around the image which controls the size of the output. If the size of the input image is $W_i \times H_i \times C_i$ where W_i, H_i, and C_i are width, height, and number of channels of the image respectively. Now, when we apply C_o filters or kernels with size $k \times k$ on the image, then the volume of the output is $W_o \times H_o \times C_o$ (Li, 2018). Where,

$$W_o = \frac{W_i - k + 2p}{s} + 1 \tag{5.5}$$

$$H_o = \frac{W_i - k + 2p}{s} + 1 \tag{5.6}$$

C_o is 3 for colour images and 1 for grayscale images. Generally, for images with C_i channels, the shape of the kernel is (k,k,C_i,C_o). The number of parameters for a kernel will be $(k \times k \times C_i + 1) \times C_o$. The CNN is mostly used with an activation function known as Rectified Linear Unit (ReLU), and the output of the CNN layer is activation maps or feature

maps. The next layer, pooling layer, in the CNN will process these activation maps (Li, 2018).

For example, consider the red channel and its respective kernel or filter with stride 1 in figure 5.3. The convolution of the image and kernel gives rise to a feature map. The first element in the feature map is calculated by using equation 5.4 as follows: $(9 \times 1) + (14 \times -1) + (6 \times 1) + (-2 \times -1) = 1$. Similarly, the rest of the elements of the feature map can be calculated.

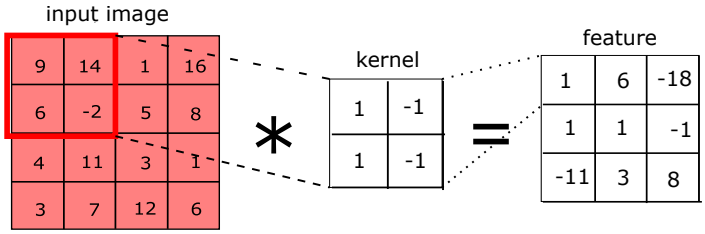


Fig. 5.3 Convolution Layer

Pooling Layer

The pooling layers reduces the dimensions of the feature maps from the convolution layers. This process is also known as sub-sampling. The dimensions are downsampled either by taking the maximum or average on patches of the image which is known as max-pooling and mean-pooling respectively. If we consider 3×3 patches over which maximum value is take to define the output layer and if the stride is 3, then the width and height of the image is divided by 3 (Rowen, 2015). This functioning is shown in figure 5.4.

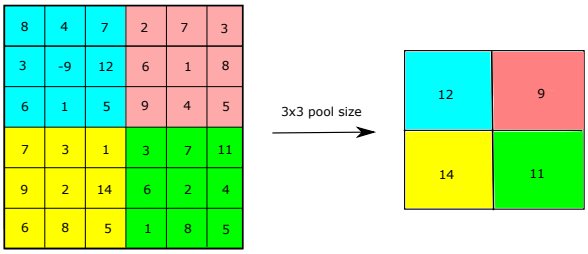


Fig. 5.4 Max-Pooling Operation

The left-hand side image in figure 5.4 is the feature map, and it is divided into four partitions, which is represented by blue, pink, yellow, and green colours respectively. Then, the max-pooling operation is applied with stride 3. During max-pooling, only the maximum value from each partition is selected to form another feature map, which is of the size 2×2 as shown in the right-hand side image of figure 5.4.

Fully Connected Layer

The role of the convolution and pooling layers is to detect high level features in an image. Once these features are detected, it is inputted to the fully connected (FC) layer as shown in figure . The FC layer takes an input volume from the output of convolution, ReLU or pooling layer. It then outputs a N dimensional vector, where N is the number of classes. For example, in a multiclassification problem with 5 classes, if the softmax approach is used, then the output of the FC layer would be in a probability vector form such as $[0, 0.1, 0.35, 0.98, 0.15]$, which indicates probability of each class. In this example, class 3 has the highest probability of 0.98, and this means the input image has a higher chance to belong to class 3 (Rowen, 2015).

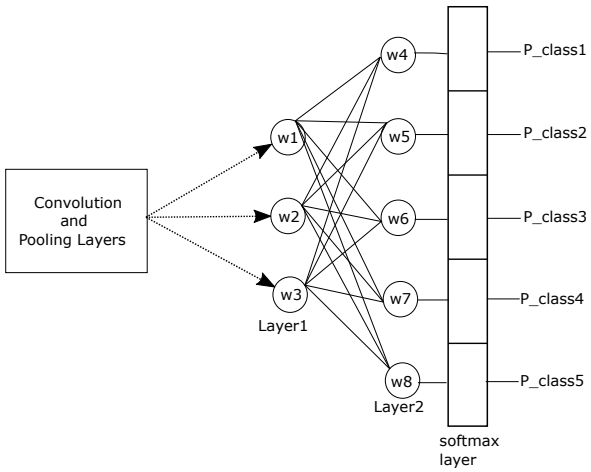


Fig. 5.5 Fully-Connected Layer

5.4 3 Layered Convolution Neural Network

In this research, the 3 Layered Convolution Neural Network (3L-CNN) has been used (Collet, 2016). Firstly, the reason for using this approach is, 3L-CNN is suited for small to medium-sized datasets which consists of few hundreds to thousands of images. The training dataset consists of 14000 images and the test dataset consists of 3500, which is considered as medium-sized dataset. Therefore, the main priority is to minimize overfitting because training the model with a lot of layers could lead to overfitting. To minimize overfitting, the entropic capacity, which is the amount of information a model can store. The number of layers in the model, higher the entropic capacity. Therefore, the 3 layers of CNN can be seen in listing 5.2.

Listing 5.1 3-Layers of CNN to Minimize Overfitting

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

If a model has a small dataset and high entropic capacity, this would lead to overfitting as the model will end up learning features from the noise and fail to generalize patterns on data that it has not seen previously. Whereas, a model with less entropic capacity will lead to the model learning less features from the data and will underfit the data, (Kalliatakis et al., 2019).

Secondly, data augmentation and dropout layers will help reduce overfitting. Data augmentation subjects the data to number of random transformations, so that a model would never see the same picture twice. This prevents overfitting and helps the model to generalize better. Dropout also minimizes overfitting by preventing a layer from seeing the same pattern again, thus acting in a way analogous to data augmentation. Both dropout and data augmentation tend to disrupt random correlations occurring in the data, (Appalaraju and Chaoji, 2017). In

listing 5.2, the final dense layer has 7 outputs this is because the dataset consists of 7 classes, and softmax layer is used in the final activation since this is a multi-classification task.

Listing 5.2 3L-CNN Dropout Layer

```
model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(7))
model.add(Activation('softmax'))
```

Another issue with the model training on a particular dataset is the selection of number of epochs and batchsize. There does not appear to be a standard practice in determining the number of epochs and batchsize. Broadly, there are two methods to select those parameters depending on the resources available. On general processing units (GPUs) hyper-parameters tuning technique is used to determine the values of epochs and batchsize Chen et al. (2016). Whereas on lower resource intensive machines running on central processing unit (CPU), a hit-and-trial method is used to determine the size of epoch and batch-size Vinayakumar et al. (2017). In this research, the latter approach is chosen because it adheres to the research methodology adopted in 3.3.

5.5 Analysis of 3L-CNN Results

The results of 3L-CNN consist of binary classification and multiclassification of memory, I/O, and CPU parameters. The multiclassification for 3L-CNN consists of 7 classes, and its results are analyzed in section . The model is trained using three datasets consisting of memory, I/O, and CPU parameters. Each dataset is further divided into train and validation dataset. The train and validation (test) dataset consists of 14000 and 3500 images for each class respectively. Firstly, the multiclassification results will be analyzed and then compared to the results of binary classification. The results of multiclassification are obtained from training and validating the datasets consisting of memory, I/O, and CPU parameters. The results from each category will be analyzed by comparing training and validation accuracies, loss, recall, and precision. As seen in section 5.4 certain measures such as dropout and augmentation

were taken to avoid overfitting, and by analyzing the results it can be concluded whether the model has overfit, underfit, or is a good fit to the data. IN this research, since the training set consists of 14000 samples, a batchsize of 20 and 50 epochs was chosen, which means the dataset is divided into 700 samples consisting of 20 batches, and each epoch contains 20 batches.

5.5.1 3L-CNN Multiclassification Analysis of Memory parameters

Firstly, the graphs of training and validation accuracy, and loss, will be analyzed to determine whether the model has overfit or underfit the data. Then, training validation and precision graphs will also be analyzed to determine how well the model has trained and validated the data.

The model has been run for 50 epochs. As seen in figure 5.6, the blue and orange curve are train and validation (test) accuracy curves respectively.

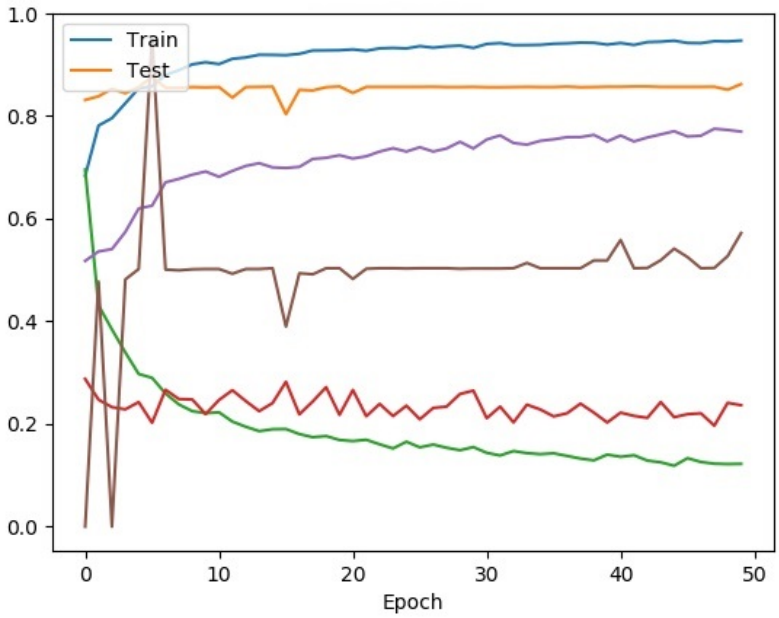


Fig. 5.6 Accuracy, Loss and Precision Curves for 3L-CNN Memory Parameters

Between epoch 1 to 6, the model underfits memory parameters as the train accuracy is less than the validation accuracy. As the number of epochs increases the train accuracy increases and the validation accuracy almost stays constant at about 0.85. This means the model has slightly overfit the data. Also the green and the red curves which are the training and validation loss curves respectively indicate the model has slightly overfit the data after nine epochs as the validation loss is greater than the training loss. When the recall of train and validation set are compared, it is seen that the recall for 3L-CNN the training curve, indicated by the grey line, fluctuates with extreme values until about eight epochs, and also decreases at the eighteenth epoch as shown in 5.7.

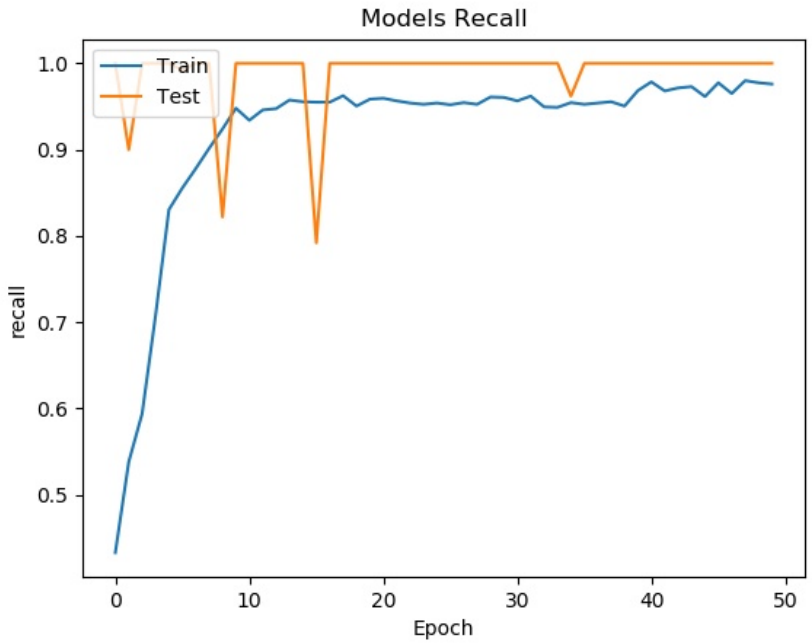


Fig. 5.7 Recall Curves for 3L-CNN Memory Parameters

Whereas, even though the recall of validation curve which is indicated by pink line has lesser values than the training curve, it is almost closer to it and it is in the higher nineties. This means the recall for both training and validation is high, which further is a sign that this model identifies almost all of the classes in the training and the validation set. But, when the precision of training and validation curves, which is shown in violet and brown curves

respectively in figure 5.6. The precision for training set varies between fifty to about seventy percent, and is fifty percent for the validation set. This means for all the patterns identified by the model, only fifty percent of them are being detected. This may be due to overfitting the data in which 3L-CNN identifies patterns of augmentation and noise as well apart from the patterns from the images. The results indicate this model fails to generalize DNA patterns when detecting them from validation set.

5.5.2 3L-CNN Multiclassification Analysis of I/O Parameters

The graphs of training and validation accuracy and loss will be compared to determine if the model is a good fit to data, or whether it has overfit or underfit the data. As seen in figurea 5.8 and 5.9, the training and validation curves are overlapping.

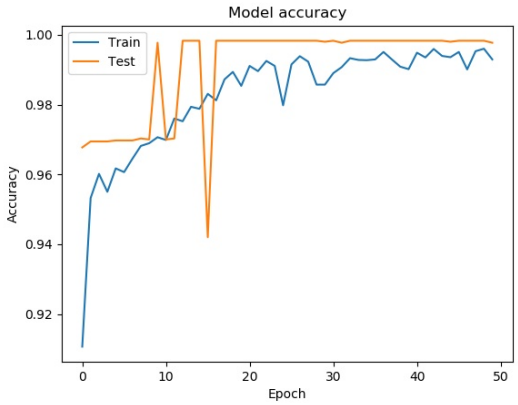


Fig. 5.8 Accuracy for 3L-CNN IO Parameters

Also, the training and validation loss curves show that the data underfits I/O parameters up to twelve epochs. But as 3L-CNN has been augmented and dropout layers added to minimize overfitting, after about twelve epochs this model handles overfitting, and hence fits data well. Also, the results of recall and precision for both training and validation sets are 1 and 0.99 respectively, which means this model identifies all images from both training and validation sets. Out of those all images that have been identified, 3L-CNN detects DNA

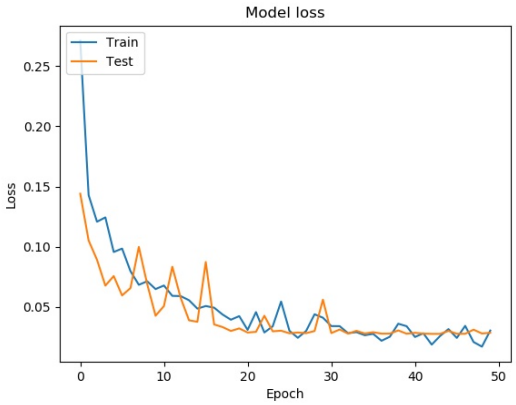


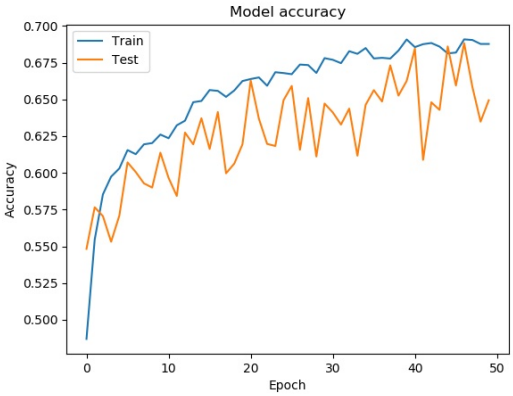
Fig. 5.9 Loss Curves for 3L-CNN IO Parameters

patterns about 99 percent of the time. Therefore, 3L-CNN fits the I/O parameters as it does not suffer from overfitting or underfitting. Hence, this model performs well in detecting DNA patterns from I/O parameters.

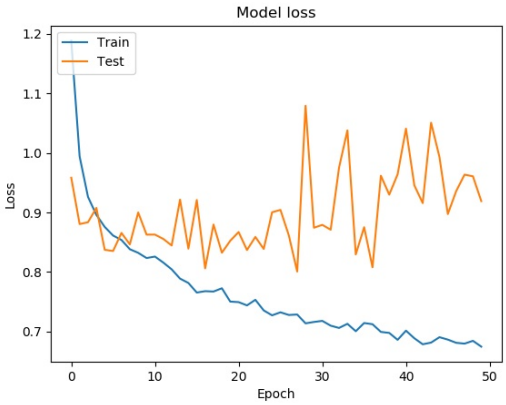
5.5.3 3L-CNN Multiclassification Analysis of CPU Parameters

Figure 5.10 shows the results of accuracy and loss for train and validation datasets. From figure 5.10a it is seen that the train curve is above or higher than that of the validation curve, but it does not give a clear picture whether this model has overfit the data as the two curves are closer to each other.

Whereas, when compared to figure 5.10b, there is a slight undefit of data until about two epochs, which is understandable as the model has still not seen all of the data from the training set. As the number of epochs gradually increases, it is seen that from eighth epoch onwards the validation curve decreases, and the training-validation curves diverge from each other. Also from figure 5.10b, it is clearly seen that the validation loss curve is greater than that of the training loss curve, which means the model has overfit the data. When the model's recall is taken into account, it can be seen that in figure 5.11a, the recall for training set is less than fifty percent and approximately fifty percent for the validation dataset.

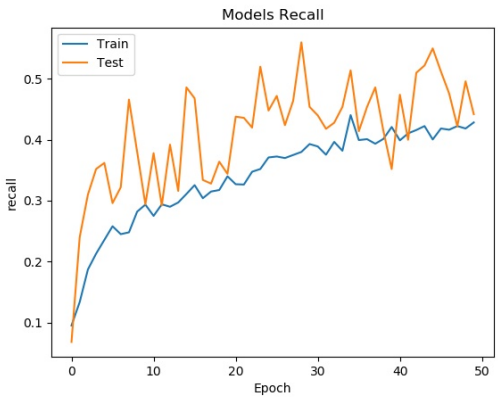


(a) Train and Validation Accuracy Curves

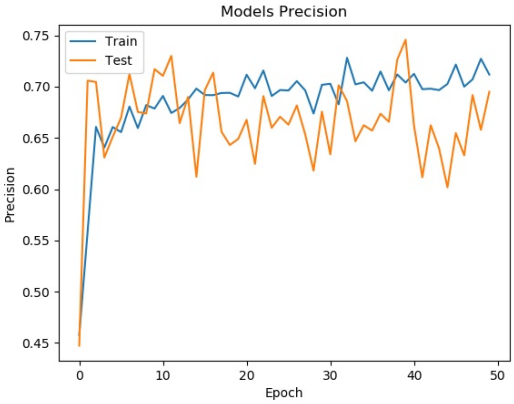


(b) Train and Validation Loss Curves

Fig. 5.10 Accuracy and Loss Curves for CPU parameters



(a) Train and Validation Recall Curves



(b) Train and Validation Precision Curves

Fig. 5.11 Recall and Precision Curves for CPU parameters

This means for CPU parameters, the 3L-CNN model only identifies less than or equal to fifty percent of the images in either of the datasets. Out of fifty percent images identified, sixty-five to seventy percent of the time the model makes the right detection as the precision varies between that range as shown in figure 5.11b which corroborates with figure 5.10 . Therefore, from figure 5.10b, the model does not perform well in detecting DNA patterns from CPU parameters.

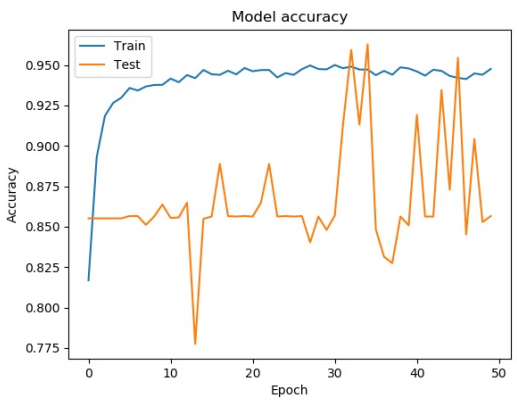
5.6 Binary Classification Analysis of 3L-CNN

Binary classification was performed to improve the detection performance. As the number of classes has been reduced from seven to two, the results shown an improvement of performance over the multiclassification approach. Especially for, memory parameters and CPU parameters. Also, in addition to train and validation accuracy, loss, recall, and precision curves even the number of false positives (FPs) and false negatives (FNs) are also analyzed for binary classification. As FPs and FNs further support the cases where the model has overfit, underfit, or fits the data well.

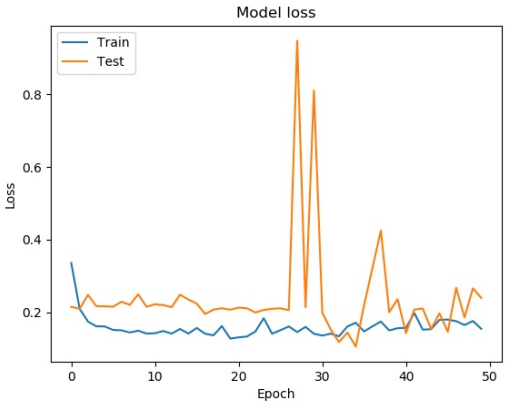
Binary classification Analysis of Memory Parameters

As seen in figure 5.12a, the train and validation accuracy curves pattern shown that the training accuracy is greater than validation accuracy up to 29 epochs. From epoch 30 onwards, the model tries to overcome or minimize the affects of overfitting. This is due to the fact that 3L-CNN has been designed to overcome overfitting by adding dropout layers and by augmenting the images. The affect of overfitting is also seen in figure 5.12b. The validation curve is greater than that of the training curve. The overfitting occurs at about the 27th and 29th epoch. From epoch 30 onwards the model tackles the overfitting issue.

Also, the average training recall rate for this model using memory parameters is 75 percent . Therefore, from the accuracy and loss curves shown in figure 5.13 and the recall and precision curves in figure , it is concluded that this model overfits the data. This means the model fails to generalize the DNA patterns by only learning noise patterns also within its training dataset. This is the reason the model has higher number of type II errors or false negatives. Hence, this model does not perform well using memory parameters.

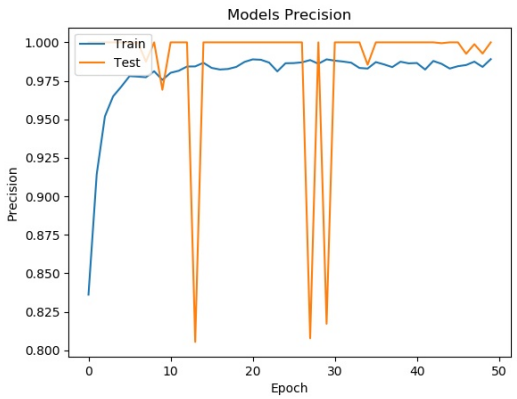


(a) Train and Validation Accuracy Curves

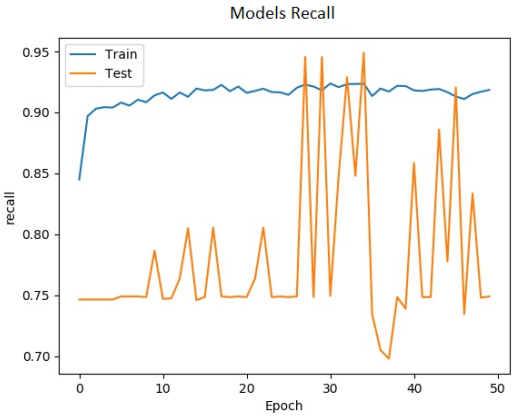


(b) Train and Validation Loss Curves

Fig. 5.12 Memory Accuracy and Loss Curves for Binary Classification



(a) Train and Validation Precision Curves



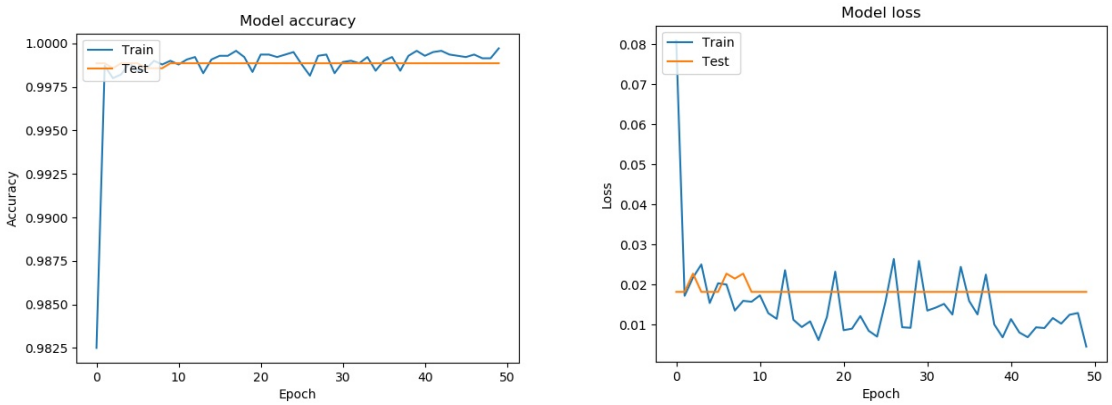
(b) Train and Validation Recall Curves

Fig. 5.13 Memory Precision and Recall Curves for Binary Classification

Binary Classification of I/O Parameters

From the accuracy and loss curves as shown in figure 5.14 it appears the model is a good fit to the data but only up to certain epochs. The training accuracy curve of training set fluctuates around the validation set up to 38 epochs as seen in figure 5.14a, and then both training accuracy increases whilst the validation accuracy remains constant. As the accuracy train curve is greater than the validation curve after epoch 38, this means the model has overfit the data. Likewise, even the training and validation loss curves as seen in figure 5.14b show

that after epoch 38, the training loss decreases and the validation loss continues to remain constant, which is also an indication that the model is overfitting the data. Therefore, it would be recommended to stop training this model at epoch 37 since the model fits the data well until that epoch.



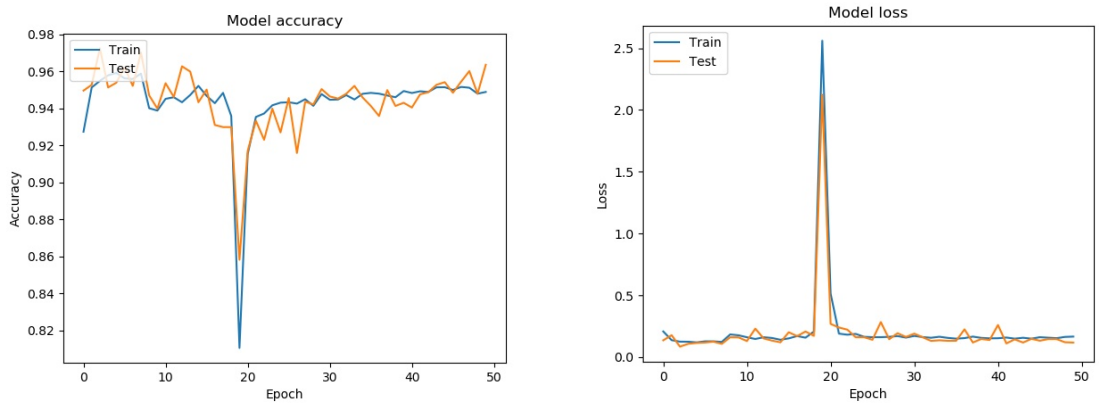
(a) Train and Validation Accuracy Curves (b) Train and Validation Loss Curves

Fig. 5.14 I/O Accuracy and Loss Curves for Binary Classification

Also, this model has hundred percent recall for both validation and train datasets, this means it is able to detect all the images in those datasets. Out of all the images detected, about 99.8 of them have their DNA patterns correctly identified. The reason for such excellent recall and precision rates is the fewer number of type I and type II errors which are four and zero respectively. This in turn means that this model works best for identifying DNA patterns in I/O parameters to detect forensic memory acquisition.

Binary Acquisition using CPU Parameters

The model's accuracy and loss curves in figure 5.15 show that the model is a good fit to the data. Although, there is a presence of a steep spike at about epoch twenty during which underfitting occurs as the training accuracy is lesser than validation accuracy, and validation loss is less than training loss. After epoch twenty, the model again fits the data well until the last epoch. The overall validation recall and precision for this model using CPU parameters



(a) Train and Validation Accuracy Curves (b) Train and Validation Loss Curves

Fig. 5.15 CPU Accuracy and Loss Curves for Binary Classification

is 98.2 and 95.57 percent. This means the model is able to identify 98.2 percent of the images in the validation dataset, and out of all the identified images the model is able to identify 95.57 percent of them correctly. Also, the reason for the spike occurring at epoch twenty for accuracy, loss, recall, and precision curves is that this model has the maximum number of false positives and false negatives during epoch twenty. After epoch twenty, the model overcomes and minimizes underfitting. Overall, this model performance is satisfactory when detecting forensic memory acquisition CPU parameters.

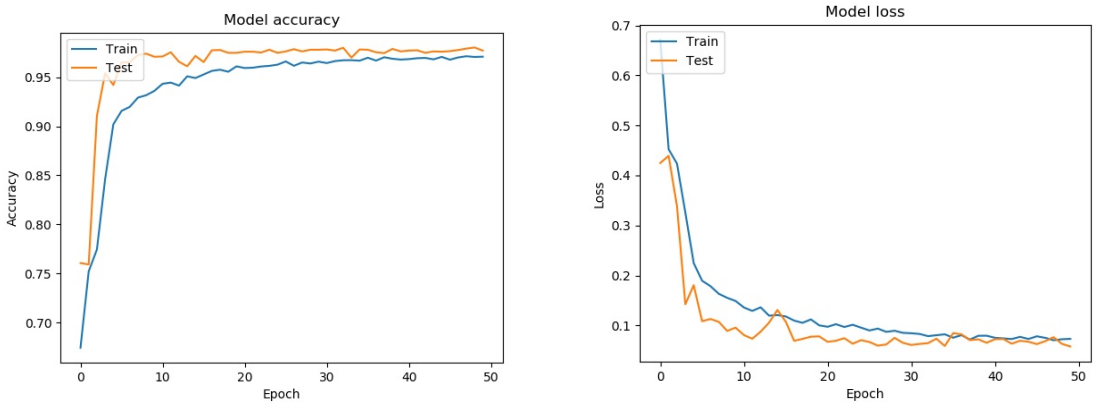
5.7 Results and Analysis of Integrated Parameters

In this section, the integrated dataset, that is, the memory, I/O, CPU (MIOC) parameters are combined into a single dataset. Then, the GAF transformation was applied to convert the MIOC dataset to images and trained using the 3L-CNN model. The datasets were multiclassified and binary classified. Multiclassification and binary classification, in essence, have the same goal, that is, to predict memory acquisition or memory non-acquisition. The difference is that multiclassification predicts not only memory acquisition or non-memory acquisition, it also classifies which forensic acquisition tool is acquiring memory

or not acquiring memory. Or, which non-forensic tool is not acquiring memory. But, the shortcoming here is that it can be difficult to maintain a higher recall and precision with fewer type I and type II errors. To, address this shortcoming, binary classification on MIOC dataset was performed, which shows slight improvement over the multiclassification strategy. The following sections discusses about multiclassification and binary classification analysis of integrated MIOC parameters.

5.7.1 Multiclassification Analysis of Integrated MIOC Parameters

The models accuracy curve shows that the validation accuracy curve is greater than that of the training accuracy curve which indicates that the model has underfit the data, but on close observation we can see that the curves are closer to each other without a big difference in values. Also, from epoch number 35, the validation and loss curves tend to be much closer to each other indicating the model is a good fit to the data. Similarly, the validation loss



(a) Train and Validation Accuracy Curves (b) Train and Validation Loss Curves

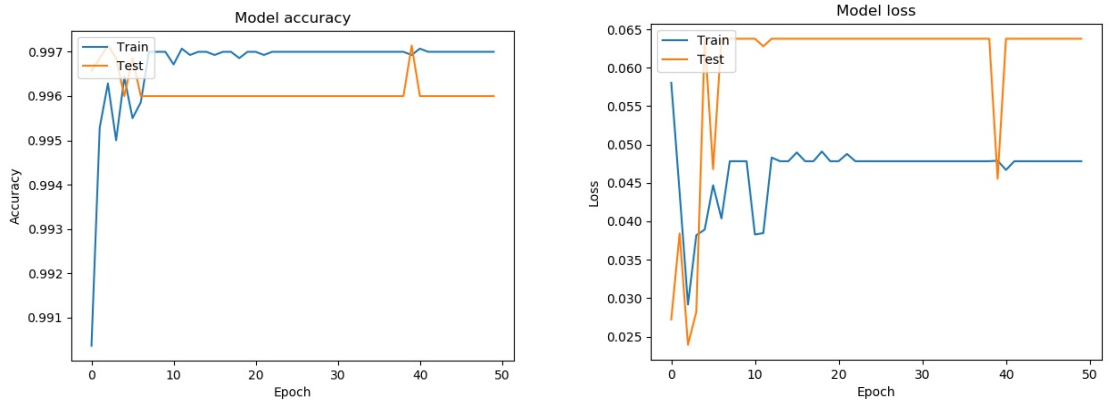
Fig. 5.16 Accuracy and Loss Curves for MIOC Multiclassification

curve is lesser than that of the training loss curve which indicates the model has underfit the data. At about epoch number 15, it can be seen that the model tries to overcome underfitting, but continues to underfit until epoch 35, after which the training and validation loss curves get very closer to each other indicating the model is a good fit to the data. Also, the overall

accuracy, precision and recall are around 99 percent in both training and validation datasets, which indicates the model generalizes the data well, and detects DNA patterns from combined MIOC parameters.

5.7.2 Binary Classification Analysis of Integrated MIOC Parameters

When the training and validation accuracy curves are first looked at 5.17, it gives the impression that the model has overfit the data because the training curve is greater than validation curve. But, with a closer observation, the graph in figure 5.17a indicates the difference between training and validation curve is just 0.001, which is not very high. Therefore, it can be argued that the model has fit the data well since the difference between training and validation accuracy curve is very low. Similarly, the training and validation loss



(a) Train and Validation Accuracy Curves (b) Train and Validation Loss Curves

Fig. 5.17 Accuracy and Loss Curves for MIOC Binary Classification

curves at first glance indicate that the model is overfitting the data. Having a closer look at the graph in figure 5.17b, it can be seen that the difference between training and validation loss curves is approximately 0.02, which is not a very high difference. Therefore, it can be argued that the model fits the data well since the difference between training and validation loss curves is very low. Also, this model has very high precision and recall which is around

99 percent, and few type I errors. Hence, this model generalizes data well, and detects DNA patterns from combined MIOC parameters.

5.8 Conclusion

In this chapter, a method to detect memory acquisition patterns was proposed. First, the dataset was transformed to images using the mathematical concept of Gramian Angular Fields. Then a 3-Layered convolution neural network was used to detect the DNA patterns. After analyzing the results, it was found that whilst analyzing the multiclassification of the parameters individually, memory parameters overfits the data whilst CPU parameters underfit the data, and therefore they are not recommended to detected memory acquisition patterns. Whilst I/O parameters is recommended as its fits the data well.

As for individual binary classification is concerned, the memory parameters is again not recommended as the model overfits the data. Whereas, the I/O parameters fit the data well but up to epoch 37. Therefore, it is recommended to stop this model at that epoch for the model to detect memory acquisition patterns accurately. The CPU parameters performance is satisfactory as it can detect about 98.2 percent patterns and out of it can correctly identify about 95.57 percent patterns that have been detected.

After analyzing the results individually using multiclassification and binary classification, the parameters were combined into a single dataset (MIOC) and later analyzed using the aforementioned classification techniques. The analysis of the muliticlassification and binary MIOC parameters shows that the model generalizes data well, and it has high precision and recall rates and fewer type I and type II errors. Hence, this shows that by combining the parameters the issues of overfitting and underfitting have been addressed to a larger extent when compared to using the model to detect memory acquisition patterns individually.

Chapter 6

A Formalism to Validate Digital Forensic Models

6.1 Introduction

The effectiveness of DNA fingerprinting technique, over signature based detection in 2.4 and various anti-forensic techniques in 2.3 has been demonstrated in chapters 4 and 5 respectively. The DNA fingerprinting technique is proposed as an anti-forensic method as it can be used to defeat the live forensic acquisition process. The research problem identified in 2.3 gave rise to research question 2 (RQ2): How can digital forensic models be validated when they are affected by anti-forensic techniques? This research question is answered in this chapter by meeting its corresponding research objective (RO6): Formalize a generic principle to validate digital forensic models by counteracting anti-forensic techniques.

Given the numerous amount of DFMs proposed in the literature, and the fact that DFMs are being enhanced by either adding or deleting a phase from the preceding models as seen in 2.1, thus making DFMs redudant. Therefore, to meet RO6, this research proposed validation principle by using mathematical formalism (Weir, 2019) to validate DFMs, instead of developing a digital forensic model or framework.

This chapter commences with the reiteration of the anti-forensic techniques that affect major phases of the forensic process, and then provides a level of abstraction for validation. After

the need for DFM validation is justfied, DFMs are validated by proving various propositions and theorems, which leads into the validation principle.

6.2 Anti-Forensic Methods Affecting DF Phases

In this section the anti-forensic methods are reviewed to find which anti-forensic methods affect which phases in the digital forensic process. The focus is on the four most common phases in the digital forensic process: Acquisition, Examination, Analysis, and Reporting.

Acquisition

When the investigator decides to acquire evidence off a machine they would follow certain procedures depending on the state of the machine i.e., whether it is ON (live acquisition) or the OFF state (dead acquisition). There are anti-forensic techniques that could defeat either of the acquisition procedures. For example, during the acquisition of memory if the drivers used in a forensic tool is dependent on the 'KDBG' string to resolve symbols, then this method could interrupt the memory acquisition process (Haruyama and Suzuki, 2012). Also, if the undocumented memory enumeration application peripheral interface (API) and memory mapping API such as MmGetPhysicalMemoryRanges and MmMapMemoryDumpMdl are patched to return a NULL value then this would return a modified version of the memory map thus resulting in incomplete acquisition of the memory as argued by Stüttgen and Cohen (2013). Malware researchers and investigators employ API call hooking techniques to study and know the behaviour of malware, but Shaid and Maarof (2015) found that this method could be prove to be futile as a malware can detect API calls and then evidence could be manipulated. Another approach undertaken by Zhang et al. (2018) is by implementing an anti-forensic technique known as Hidden in I/O Space (HIveS) and malicious enclave software to tamper with the acquisition process to prevent memory analysis. This technique is designed to operate outside the scope of the operating system and therefore making it

harder to detect its presence in the memory. Even hard drive acquisition can be defeated if the attacker adopts techniques such as anti-forensics of data storage by alternative use of communication channels (AFAUC) (Baier and Knauer, 2014). In AFAUC, the storage device is accessed through its diagnostic interface to hide or even obfuscate data such that it will not be accessible to the investigator. Moreover, this hidden data will also be absent in hidden areas such as host protected area and device configuration overlay of the hard drive which the investigator might look into before making a forensic image of the device. Hence, defeating the forensic acquisition process.

Examination

This phase of the forensic process can be mainly defeated by encrypting the whole storage media or just certain partitions within the storage media. Or, even applying steganography techniques such as hiding a file system within a file system which can be accomplished by the encryption tool TrueCrypt. This is a useful technique for an attacker because they can disclose the encryption key or passphrase for the first encrypted file system, but not for the other hidden filesystem which could result in the investigators being unaware of the hidden filesystem. Due to this clever technique the suspects in the UK would be able to comply with section 49 of the Regulation of Investigatory Powers Act 2000 (RIPA 2000), but they would be able to avoid punishment mentioned in section 53 of the RIPA, thus deceiving the investigation and the legislation used to tackle encryption key issues. Other ways of hiding a file to prevent examination as mentioned by Dahbur and Mohammad (2011) is to manipulate certain registry key, hiding data in the HPA and DCO areas of the hard drive, or even using bootable USBs or DVDs, and compression bombs such as 42.zip.

Analysis

During the analysis phase the investigation certain files or areas of the storage device will be looked into to find relevant information pertaining to the case. This process can be defeated or prolonged depending on the anti-forensic technique. A kernel rootkit disk filter driver known as 'Ddefy.sys' can be used to hide a file from a New Technology File System (NTFS) filesystem. The driver can locate the filename, its directory entry position, the clusters containing data, and its disk position and with this information the data can be hidden. The same rootkit can also defeat memory analysis by performing a system service dispatch table (SSDT) hook which can be used to modify a process and thread list Bilby (2006). Also, encryption of certain files with strong passphrases will consume lot of the investigation time, steganography techniques of hiding a file within a file may cause the investigator to overlook certain important files. Tools such as metasploit's slacker which can be used to delete data in the slack space which would make it impossible to recover deleted data, and metasploit's transmogrify can be used to change metadata such as modified, accessed, created parameters of a file thus defeating timeline analysis as argued by Garfinkel (2006), Dahbur and Mohammad (2011), and Gül and Kugu (2017).

Reporting

It is possible to inject malicious code into computer forensic tools (CFTs) to produce false forensic reports and also infect the computer. This kind of attack can be done by adopting hypertext markup language (HTML) code injection technique wherein a malware can be embedded into a report that can be used to attack web browsers. Now, when the report is viewed in to web browser the malware escapes the virtual environment and infects the machine which was demonstrated by Wundram et al. (2013).

After reviewing various anti-forensic techniques,the findings show that various phases in the digital forensic process are subjected to anti-forensic attacks as shown in Table 6.1.

Collection	Examination	Analysis	Reporting
AFAUC	Data Hiding	Steganography	Code Injection
Kernel Debugger Block Hiding	Artefact Wiping	Data Manipulation	
Memory enumeration API Hooking	Cryptography	Code Injection	
Memory Mapping API hooking	Encryption	Adding known files	
Data Pooling	Data manipulation	String decoration	
	AFAUC	False Audit Trails	
	Compression Bombs	Hash Collisions	
	Denial of Service Attack	Loop References	
	ReDOS	Restricted Filenames	
	Transmogrification		

Table 6.1 AF techniques affecting various digital forensic phases

6.3 Abstracted Digital Forensic Framework

In the literature of digital forensics (DF) numerous digital forensic models were reviewed in section 2.1. One of the important aspects of DF research is tool testing. The National Institute of Standards and Technology (NIST) oversees this aspect of DF research and various DF tools are tested and then reported on their website National Institute of Standards and Technology. Then there is FSR-G-218 method validation in digital forensics in the United Kingdom (UK) and Daubert Standard in the United States (US) to validate digital forensic methods and procedures. In this research it is argued that DFMs, DF tool testing, DF Method validation, and the concerned legislation are inter-related and have their layer of abstraction. An abstracted framework is proposed that shows the inter-relationship among the first three layers and their relationship to the fourth layer. The four layers of abstraction are described below:

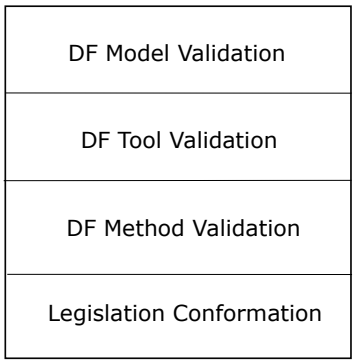


Fig. 6.1 Levels of Abstraction for Testing Digital Forensic Models/Frameworks

6.3.1 DFM Validation

The first layer is the DFM. The investigation of digital crime is a step-by-step process. This process can be guided by a standard operating procedure (SOP) or a DFM as to how to go by collecting, preserving, analysing and reporting digital evidence. Now, the procedure to collect digital evidence will be guided by a method. For example, during the collection phase if a computer is the ON state then specific procedures will be followed to gather relevant evidence Williams (2012). And when gathering digital evidence off a computer, this will entail a forensic software tool. During this process of collecting evidence, care will be taken to maintain the integrity of the evidence and this is done so as to conform to the legislation. Now, to maintain evidence integrity the method and the forensic tools used have to be validated. Method validation and tool validation are discussed below.

6.3.2 Tool Validation

Ensuring reliability of digital forensic tools is of prime importance to the forensic community. The National Institute of Standards and Technology (NIST) in the United States (US) set up the Computer Forensic Tool Testing Program (CFTT) for this purpose National Institute of Standards and Technology. The CFTT projects evaluates digital forensic tools by adopting functional conformance testing i.e., a set of requirements are established by developing a set

of test assertions and cases. And set of setting procedures are used to perform tests. Over the years numerous digital forensic tools have been tested and documented on their project website. As of October 2017 onwards in the UK, the FSR in its code of practice made it mandatory that all providers of digital forensic services to the criminal justice system must be accredited to ISO-17025, which provides general requirements for the competence of testing and calibration laboratories.

6.3.3 Method Validation

In the US a method or procedure adopted by a investigator has to be adhered to the Daubert Standard, which provides a set of objective guidelines for judges to determine the admissibility of scientific evidence in the courts. The Daubert standard applies to those digital forensic methods or procedures that were used to uncover evidence from digital devices, and must satisfy the following criteria Meyers and Rogers (2005)

1. "Testing: Can and has the scientific procedure been independently tested? Peer Review: Has the scientific procedure been published and subject to peer review?"

2. "Error rate: Is there a known error rate, or potential to know the error rate, associated with the use of this scientific procedure?

3. Standards: Are there standards and protocols for the execution of the methodology?"

4. "Acceptance: Is the scientific procedure generally accepted by the relevant scientific community?"

In the United Kingdom (UK), in 2005 the House of Commons Science Technology Committee (2019) in the paragraph 173 mentioned that validation of scientific methods are absent before they are admitted in the court, and judges are incompetent to determine the validity of scientific evidence and they recommended Forensic Science Advisory Council

which belongs to Forensic Science Regulator. (2019) to develop a test for scientific evidence which should build on the Daubert Test. In 2016, the Forensic Science Regulator (FSR) has produced a guidance on method validation in digital forensics FSR-G-218 as seen on Forensic Science Regulator (2016). It amalgamates essential information from International Standards Organisation (2017), FSR-G-201 validation guidance found in Forensic Science Regulator (2014), Scientific Working Group on Digital Evidence (2019), and Criminal Practice Directions as mentioned in Courts and Tribunals Judiciary (2014).

6.3.4 Legislation Conformation

If the investigator's methods, procedures, tools, process are validated, then the chances of the evidence being accepted in the court increases. For example, if the investigation method fits or satisfies the criteria and objectives of the daubert standard then it would be accepted or admissible under the Federal Rules of Evidence (FRE) 702 and 902 (Upcounsel, 2019). In the abstraction layers the digital forensic tools and methods can be validated, but a procedure to validate DFMs is not present in the literature. In the next section, why validating DFMs is important is justified, and in section 6.5 a principle to validate DFMs is proposed.

6.4 Justifying the Validation of DFMs

In section 6.1.2 of FSR-G-218 it is mentioned that risk assessment must be conducted before performing validation testing. Risk assessment in the Criminal Justice System (CJS) usually includes: "a. the risk of wrongful conviction(s); b. the risk of wrongful acquittal(s); and c. the risk of obstructing or delaying investigation(s)."

The aforementioned three circumstances can occur if one is not aware of the anti-forensic methodologies and techniques as pointed out by Dahbur and Mohammad (2011), and Garfinkel (2006). Apart from this, if the anti-forensic techniques/methodologies as shown in

Table 6.1 is not taken into consideration into the risk assessement procedures of FSR-G-218 or any guidance document that depends on it, then this can pose a threat to the investigation as far as reliability of evidence is concerned which is specifically mentioned in the direction 19A.6 of the Criminal Practice Directions.

Previously, according to section 69 of Police and Criminal Evidence Act 1984 (The National Archives, 2019), it was mandatory to prove that the computer was functioning properly, and was not wrongly used to produce a document that could be admitted as evidence. This rule has been repealed by section 60 of the Youth Justice and Criminal Evidence Act 1999 (The National Archives., 2019). And now, the computer evidence adheres to the common law rule that a presumption exists that the computer producing evidence was functioning properly at the time of investigation. Therefore, the evidence is admissible in the court of law. However, this presumption can be rebutted if either of the parties (prosecution or defense) adduce evidence to the contrary. Digital Forensic Investigators often use forensic software tools to generate forensic reports. Wundram et al. (2013) demonstrated an anti-forensic technique that attacks a forensic tool to generate false forensic reports. If either of the party proves that the investigation was compromised, then according to rebuttable presumption the evidence might be adjudged inadmissible in the court of law. Most of the digital forensic models in section 2.1 or the FSR-G-218 guidance on method validation in digital forensics do not address anti-forensic issues. This research addresses the aforementioned problems by proposing a principle for validating DFMs.

6.5 Validating Digital Forensic Models

In this section, a principle for validating DFMs by counteracting anti-forensic techniques that affects each phase in the digital forensic process is proposed. The logic behind this principle is that for a DFM to be validated, every phase in the DFM must be validated before proceeding to the next phase, and when all the phases are validated then it can be concluded

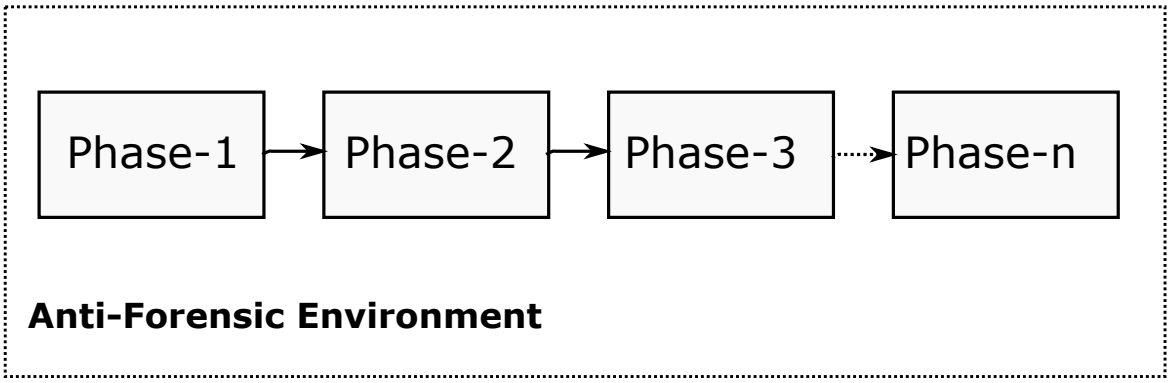


Fig. 6.2 n-phase DFM in an AF environment

that a DFM is validated. For a DFM phase to be validated, the anti-forensic techniques in each and every phase of the digital forensic process must be accounted for, i.e., the AF techniques should be detected and countered. The process of detecting and countermeasuring anti-forensic (AF) techniques in each phase of the digital forensic process is the essence of the validation principle and is formalized using the concept of tensor products.

Consider a DFM with n phases in an anti-forensic environment as shown in Figure 6.2.

According to the validation principle, firstly individual phases need to be validated in their own AF environment. Now consider Phase 1 in Figure 6.2 in its anti-forensic environment. Its AF environment is defined by its respective AF techniques, for example those shown in Table 6.1. The first and the foremost step is to detect if an anti-forensic technique or method is acting upon or affecting a phase in the digital forensic process or not. To mathematically express this process certain definitions and axioms are first proposed, and then propositions are proved with the help of these definitions and axioms. Mathematical notations are presented in table 6.2.

Let the anti-forensic methods or techniques (A) acting upon a DF phase be represented by the vector $A = \{a_1, a_2, a_3, \ldots, a_n\}$ such that $a_i \in A$, and the detection methods be represented as $D = \{d_1, d_2, d_3, \ldots, d_n\}$ such that $d_i \in D$.

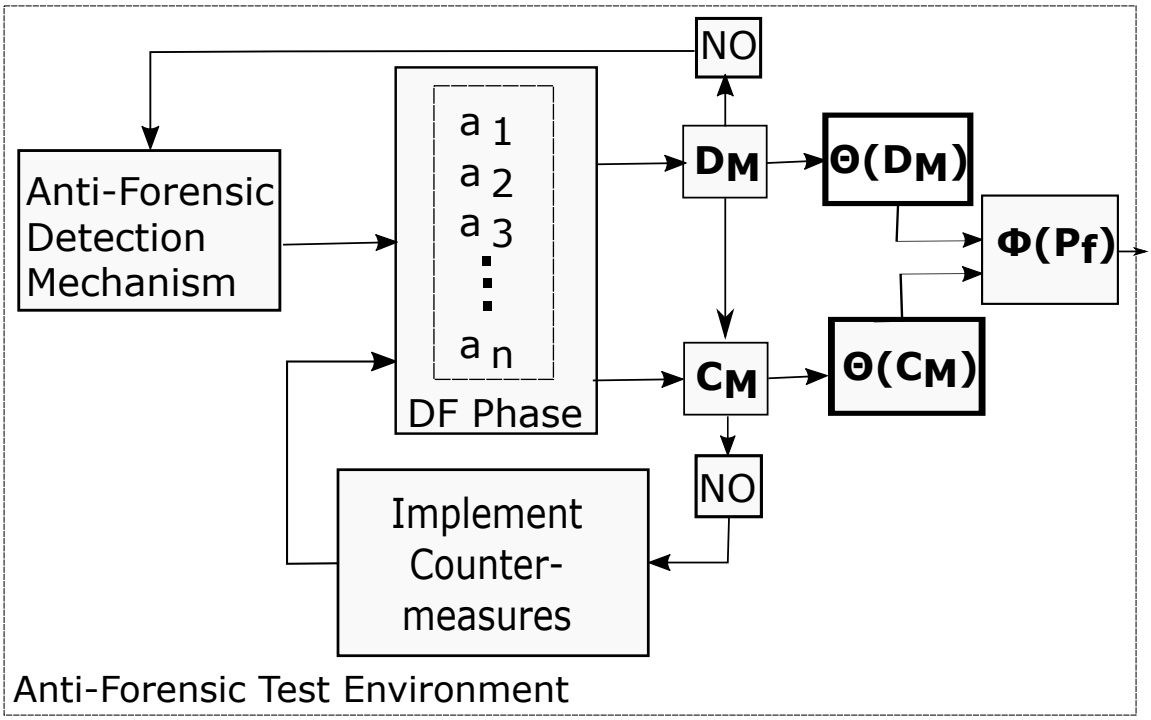


Fig. 6.3 DF Phase Validation

Definition 6.5.1 *The logical detector product (D_M) is defined as $\{a_j.d_j \in D_M | D_M = a_j \wedge d_j{}^T\}$, which means an anti-forensic technique ($a_j \in A$) can be detected in 'n' unique number of ways using elements in D. This is analogous to inputting the individual vectors* ***a*** *and* ***d*** *to logical and-gates.*

Definition 6.5.2 *The counter tensor product (C_M) is defined as $\{e_j.c_j \in C_M | C_M = e_j.c_j{}^T\}$. The significance of D_M and C_M as shown in Figure 6.3 is that it shows us if an anti-forensic technique is detected or countered i.e., if the logical and of $a_j.d_j = 1$ then a_j is detected. Likewise, a_j is countered if $e_j.c_j = 1$.*

The relevance of DF phase validation in the context of forensic investigation is that, it ensures a particular phase in the DFM is under the affect of an AF attack or not. If the DF phase is being influenced by an AF tool or procedure, then the DF validation procedure could

detect such an attack and counter it. And, if a phase is free of AF attacks, then this decreases the chance of making wrong conclusions during the course of an investigation.

6.5.1 Vector Transformation Operators

The vector transformation operators play an important role in transforming D_M and C_M. If $m_{ij} \in \mathbb{R}^{p\times q}$ is a Matrix M, then $\Theta^n_{r,k}(M_{p\times q}) \longmapsto N_{p\times 1}$, which means $\Theta^n_{r,k}$ on $M_{p\times q}$ yields $n_{ij} \in \mathbb{R}^{p\times 1}$ matrix N a column vector of order $p \times 1$. Here, the subscript r means the transformation is only being applied to all the rows in M, and 'k' and 'n' means the transformation is from row 'k' to row 'n'. If only a specific row 'i' is considered in M, then the row vector transformation on row j in M is defined as logical-or of all elements in row j, i.e., $\Theta_{r,i}(R_i) = \mathbb{V}^n_{i=1}(r_i)$, and this operation yields a singleton set by transforming the row vector R_i of order $1 \times q$ to a -matrix of order 1×1. Now, if the row vector transformation is applied to all rows in M, beginning from the first row to the nth row, then $\Theta^n_{r,1} = \{\Theta_{r,1}(R_1), \Theta_{r,2}(R_2), \ldots, \Theta_{r,n}(R_n)\}$, and this transformation yields a matrix N of order p$\times$1, i.e., $\Theta^n_{r,1}(M_{p\times q}) = N_{p\times 1}$. Similarly, $\Omega^n_{r,k}(M) = \wedge^n_{i=1}(r_i)$ and
$\mathscr{Z}^n_{r,k}(M) = \Sigma^n_{i=1}(r_i)$ are logical & , and summation over each row in Matrix M respectively.

6.5.2 Formalization of DFM Validation

The notions of detecting AF attacks in DFM phases (Propositions 6.5.1 and 6.5.2) and of validating DFMs (Theorem 6.5.1) are formalized in this section. Before moving onto explaining the propositions, it is important to understand the function of the elements in Figure 6.3, which is the diagrammatic representation of DFM phase validation. Now, if a phase in a DFM is considered, say the first phase, then this phase must be accounted for anti-forensic attacks. This is done with the help of the elements D_M and C_M. In order to detect AF attacks, the AF detection mechanism may consist of numerous techniques or methods to detect a single or numerous AF technique(s). These detection methods and AF

techniques form a matrix (D_M). And when certain operations are applied to D_M, certain results can be deduced from those operations, such as, when do AF techniques are said to be detected, not detected, countered or not countered. This operation is mathematically formalised and proved in proposition 6.5.1.

Proposition 6.5.1 *(a) If in a DFM phase,* $\Omega_{c,1}[\Theta^n_{r,1}(D_M)] = [k]_{1\times 1}$, $k \in \mathbf{B}$, *and if k=1 then the AF techniques* $a \in A$ *are detected, and if k=0, then A is undetected.*
(b) If $\Omega_{c,1}[\Theta^n_{r,1}(C_M)] = [k]_{1\times 1}$ *then the detected AF technique* $a \in A$ *is countered if k=1, and is not countered if k=0.*

Proof: Consider any arbitrary DF phase, and if $a \in A$ are the n anti-forensic vectors acting upon the phase and if each vector in A can be detected by 'n' number of ways, then the detector tensor product of A and D is defined as $D_M = a_j.d_i^T$, where $a_j \in A$, $d_i \in D$. Let's assume D_M is a square matrix of order $n \times n$ consisting of 'n' number of rows defined by the set $R_n = \{R_1, R_r, \ldots, R_n\}$, where any arbitrary row $R_j$row in R_n is defined as $\{R_j = a_j.d_1, a_j.d_2, \ldots, a_j.d_n\}$. If $\{\exists a_j.d_i \in R_j | a_j.d_i = 1\}\ \forall R_n$ then a_j is detected by d_i. Now, if a row vector transformation is applied on D_M, then it yields an all ones column vector (E) of order $n \times 1$ i.e., $\Theta^n_{r,1}(D_M)] = E_{n\times 1}$. Therefore, the column transformation of E, $\Omega_{c,1}(E) = \Omega_{c,1}[\Theta^n_{r,1}(D_M)] = [1]_{1\times 1}$, therefore $\forall a \in$ A is detected in a DFM phase. Now, if $\{\exists a_j.d_i \in R_j | aj.d_i = 0\}\ \forall R_n$ then $\Omega_{c,1}[\Theta^n_{r,1}(D_M)] = [0]_{1\times 1}$, and the DFM phase is still under the effect of an anti-forensic attack. Proposition 1(a) is also known as the sufficiency principle. After the anti-forensic technique a_j is detected, it is passed to the counter product (C_M) as shown in Figure 6.3. The detected anti-forensic methods $a_j \in A$ in D_M can be countered by C=$\{c_1, c_2, c_3, \ldots, c_j, \ldots, c_n\}$ in 'n' number of ways, and thus the counter product is defined as $C_M = (a_j.d_j).c_j^T = (e_j).c_j^T = E.C^T$. Now in C_M if there exist at least one c_j such that axiom of counterability is satisfied then a_j is countered by c_j i.e., if $\{\exists e_j.c_i \in R_j | e_j.c_i = 1\}\ \forall R_n$ then a_j is countered by c_i, and if all the values on R_n are 0 then the anti-forensic technique is not countered i.e., if $\{\forall e_j.c_i \in R_j | e_j.c_i = 0\}\ \forall R_n$ then a_j is not countered by c_i.

The purpose of D_M and C_M is not only to notify whether AF techniques are detected and countered as proved in proposition 6.5.1, but also one can specifically pin-point as to which detection method(s) has detected an AF attack(s). This is stated and proved in proposition 6.5.2.

Proposition 6.5.2 *If* $[\Theta_{r,i}(a_j.d_i)] = [1]_{1\times 1}$ *then* a_j *[* $\pi_{i=0}^{N-1-i}(\bar{S_i})\ S_{N-i}\ \pi_{k=0}^{i}(\bar{S_{N-K}})\ S_{N-i}$*] gives the element* $d_i \in R_i$ *that detects* a_j.

Proof: Since detector product D_M is defined as $\{a_j.d_j \in D_M | D_M = a_j.d_j{}^T\} = \{R_j \in R_n | R_n = a_j.d_n\}$. Therefore, D_M is a set of rows R_1 to R_n, that is, $\{R_1, R_2, R_3, \ldots, R_n\}$. Now consider any arbitrary row $R_j \in R_n$ defined as R_j=$a_j.d_1 + a_j.d_2 + a_j.d_3 + \cdots + a_j.d_n$. Now, if $a_j \in A$ forms the inputs to a n:1 multiplexer, and d forms the select lines $S = \{S_i \in S_n | S_i = (\pi_{i=0}^{N-1-i}\bar{S_i}.S_{N-i})(\pi_{k=0}^{i}\bar{S_{N-k}})\}$. Now if d_i and S are bijective, i.e., f:$d_i \rightarrow$ S. This can be represented as

$$\begin{pmatrix} d_0 \\ d_1 \\ \ldots \\ d_{n-1} \\ d_n \end{pmatrix} \rightarrow \begin{pmatrix} \bar{S_0} & \bar{S_1} & \bar{S_2} & \ldots & \bar{S_{N-1}} & S_N \\ \bar{S_0} & \bar{S_1} & \bar{S_2} & \ldots & S_{N-1} & \bar{S_N} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \bar{S_0} & S_1 & \bar{S_2} & \ldots & \bar{S_{N-1}} & \bar{S_N} \\ S_1 & \bar{S_0} & \bar{S_2} & \ldots & \bar{S_{N-1}} & \bar{S_N} \end{pmatrix}$$

Therefore, R_j=$a_j(\bar{S_0}\bar{S_1}\ldots S_N) + \cdots + a_j(S_0\bar{S_1}\ldots\bar{S_N})$. This equation shows what d_i in R_j detects a_j. For example, if R_1=$a_j(\bar{S_0}\bar{S_1}\ldots S_N)$, this means a_1 is detected by d_1 in row 1 (R_1) of D_M.

Now. after proposition 6.5.1 is satisfied it is imperative to calculate the pass function P_f, which indicates to the investigator whether the phase has been validated, and hence the process can be proceeded to the next stage. P_f addresses the issue of assuming that it is safe to proceed from one phase of the DFM without considering the effects of AF as seen in the finding of the review in Section **??**. And, when all phases have been validated, then the validation principle is satisfied. The validation principle is stated in theorem 6.5.1.

Table 6.2 Mathematical notations

Maths Notation	**Meaning**
D_M	Detectability Matrix
C_M	Counterability Matrix
P_f	Phase Function
$\langle a_j.d_i \mid a_j \times d_i \rangle$	Dot Product or Cross Product
$\Theta^n_{r,1}(M)$	Vector−Or Transformation
$\Omega^n_{r,1}(M)$	Vector−And Transformation
${}_{r,1}\Phi^n_{c(r,1)}(M)$	Vector And−Or Transformation

Theorem 6.5.1 *If (a)* $\{\forall P_i \in P_n | P_i = 1 \vee {}_{r,1}\Phi^n_{c(r,1)}(P_n) = 1\}$ *then the DFM is validated. (b)* $\{\exists P_i \in P_n | P_i = 0 \vee {}_{r,1}\Phi^n_{c(r,1)}(P_n) = 0\}$ *then the DFM is invalidated.*

Proof: By the definition of validation principle a DF phase is validated if all the anti-forensic techniques of the respective phases in a DFM are detected and countered. Mathematically, the validation principle is defined as

$P_{fj} = [\Omega_{c,1}[\Theta^n_{r,1}(a_j.d_i)]].\ [\Omega_{c,1}[\Theta^n_{r,1}((a_j.d_i).c_i)]] =$

$\Omega_{c,1}[\Theta^n_{r,1}(a_j.d_i).((a_j.d_i).c_i)] = \Omega_{c,1}[\Theta^n_{r,1}(a_j.d_i).(e_j.c_i)] =$

$\Omega_{c,1}[\Theta^n_{r,1}(D_M).(C_M)] \Rightarrow \Omega_{c,1}[\Theta^n_{r,1}(P_i)]$.

Also, according to the validation principle if the phase function of the individual phases P_i and $i \in N$ in DFM is $[1]_{1\times 1}$, then its respective phases are validated. Consider a DFM with its respective phase functions as shown in Figure 6.4. Now, the individual phase functions is P_n=$\{P_1, P_2, \ldots, P_n\} \Rightarrow \{\Omega_{c,1}[\Theta^n_{r,1}(P_1)], \Omega_{c,1}[\Theta^n_{r,1}(P_2)], \ldots, \Omega_{c,1}[\Theta^n_{r,1}(P_n)]\}$, which is a row vector and represented as ${}_{r,1}\Phi^n_{c(r,1)} = \Omega_{r,1}[\Omega_{c,1}[\Theta^n_{r,1}(P_i)]]$ and after the application of proposition 6.5.1 (a) and (b) to ${}_{r,1}\Phi^n_{c(r,1)}$ if all phase functions in a digital forensic model is $[1]_{1x1}$ i.e., $\{\forall P_i \in P_n | P_i = 1\}$, then ${}_{r,1}\Phi^n_{c(r,1)}(P_n)$ =1 and the DFM is validated. Else if at least one of the elements in P_n or if at least one of phase functions in a digital forensic model is 0 i.e., $\{\exists P_i \in P_n | P_i = 0\}$, then ${}_{r,1}\Phi^n_{c(r,1)}(P_n)$=0 and the DFM is invalidated.

The forensic implication of DF phase functions, is that if even one of the phases is under the influence of an AF attack, then the investigation must not be proceeded any further. This

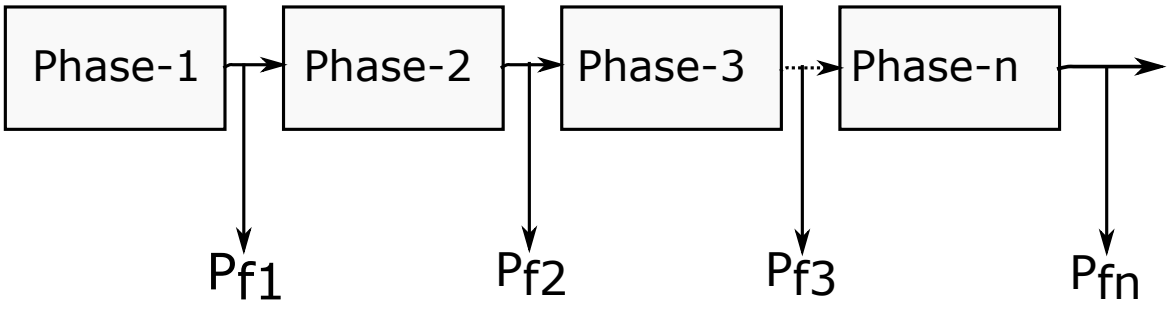


Fig. 6.4 DF Phase Functions

would enable the investigator indentify and rectify the AF threat. Therefore, avoiding any cases that would defeat the purpose of digital forensic process. If and only if all the phases in a DFM are not under the influence of an AF attack, only then the investigator could proceed further with the investigation.

6.6 Evaluation

This section demonstrates the feasibility of the validation principle by evaluating three DFMs. The first DFM is based on a hypothetical scenario to show a DFM that is successfully validated, and the other two DFMs selected from the literature are Rekhis and Boudriga (2012b) Rani and Kumari (2017), to prove how these DFMs fail validation using the validation principle. All DFMs are evaluated based on the validation principle as mentioned in section 6.5. Firstly, a hypothetical situation is explained, and then the evaluation process of the DFM phases is processed in four steps: 1) The AF column vector **a** and the detection row vector **d** are determined. 2) The vector-or transformation $\Theta(D_M)$ is computed to detect anti-forensic techniques in a DF phase. 3) The vector-or transformation $\Theta(C_M)$ is computed to counter anti-forensic techniques in a DF Phase. 4) Then finally, the phase function P_f is computed. And based on the value of P_f validate or invalidate a DFM.

6.6.1 Validation Case

The application of this principle is shown by using a hypothetical situation where in a defendant is accused of DDoS attacking an organisation's network from their laptop, and the defendant claims that they're unaware of it and that their laptop might be hacked to perpetrate the crime. Now, let us assume the investigator follows a SOP and applies the principle to test for the validation of the DFM (before proceeding with the investigation and making a report to the court) which comprises of 4 phases collection, examination, analysis, and reporting. The objectives for each phase are as follows:

1. Collection: To collect or gather evidence from physical memory and hard drive.
2. Examination: To examine the content of the hard drive such as what is the size of the hard drive, volumes, hidden files, file types etc.
3. Analysis: To answer questions pertaining to investigation or support the investigative hypothesis/counter-hypothesis.
4. Report: To make a report of the findings.

To keep this case study simple, it is assumed that each phase in the DFM has two AF techniques a_1 and a_2 (except for reporting phase), and a_1 and a_2 have only d_1 and d_2 as their detecting methods and C_1 and C_2 as their countermeasures respectively.

Collection

During the collection phase the investigator uses a forensic tool to image the physical memory, and since the principle is being used the AF techniques in the collection phase (a_1 and a_2) will be looked up. Let us assume the AF techniques "Memory enumeration API hooking" and "Memory Mapping API hooking" is acting up on the collection phase, and also assume these two AF techniques can be detected and countered. The following four steps shows the application of the principle to validate the collection phase:

Step1: Determine column vectors **a** and **d**. The two AF techniques affecting the collection phase are:

a_1 = Memory enumeration API hooking

a_2 = Memory Mapping API hooking, and

$d_1 = d_2$ = A forensic tool that can detect a_1 and a_2.

Step 2: Compute the detector matrix once the vectors **a** and **d** are determined. By definition, $D_M = \begin{pmatrix} a_1.d_1 & a_2.d_2 \end{pmatrix}$. Since, for a_1 can be detected by d_1, and a_2 can be detected by d_2. Therefore, the logical *and* operation of $a_1.d_1 = 1$, and $a_2.d_2 = 1$. This implies, $D_M = \begin{pmatrix} 1 & 1 \end{pmatrix}$. Therefore, $\Omega_{c,1}[\Theta^n_{r,1}(D_M)] = [1]_{1\times 1}$

Step 3: Compute C_M once the AF techniques a_1 and a_2 have been detected. Now, a_1 and a_2 can be countered by, $c_1 = c_2$ = DumpIt, a memory acquisition tool resistant to a_1 and a_2, which means $ap_2.c_2$ = logical 1. And, by definition $C_M = \begin{pmatrix} ap_1.c_1 & ap_2.c_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 \end{pmatrix}$. Therefore, $\Omega_{c,1}[\Theta^n_{r,1}(C_M)] = [1]_{1\times 1}$

Step 4: Compute phase function (P_f) once the values of D_M and C_M have been determined. Therefore, $P_{fCollection} = \begin{pmatrix} 1 & 1 \end{pmatrix} \Rightarrow \Omega_{c,1}[\Theta^n_{r,1}(P_1)] = 1$ the collection phase is validated since proposition 6.5.1 is satisfied, and the investigator can proceed to the next phase, Examination.

Examination

The process of validating the examination phase is shown in the following four steps:

Step1: Determine column vectors **a** and **d**. The two AF techniques affecting the examination phase are:

a_1 = Encryption

a_2 = Compression Bombs

d_1 = Forensic Software (example EnCase 8)

d_2 = Intelligent Decompression Libraries

Therefore, $a_1.d_1 = 1$ and $a_2.d_2 = 1$, since a_1 and a_2 are detected by d_1 and d_2

Step 2: After determining the values of **a** and **d**, compute the detector matrix D_M. By definition, $D_M = \begin{pmatrix} a_1.d_1 & a_2.d_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 \end{pmatrix}$. Therefore, $\Omega_{c,1}[\Theta^n_{r,1}(D_M)] = [1]_{1\times 1}$

Step 3: Compute C_M once the AF techniques a_1 and a_2 have been detected, that is, D_M has been determined. Let's say a_1 can be countered by c_1 = Encryption Key, and $c_2 = 1$, since it has been detected in step 2 (see special case in section III). Therefore, $ap_1.c_1$ = logical 1 and $ap_2.c_2$ = logical 1. By definition, $C_M = \begin{pmatrix} ap_1.c_1 & ap_2.c_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 \end{pmatrix}$. Therefore, $\Omega_{c,1}[\Theta^n_{r,1}(C_M)] = [1]_{1\times 1}$

Step 4: Compute phase function (P_f) once D_M and C_M for the examination phase are determined. From steps 2 and 3, $P_{f_{Examination}} = [1\ 1] \Rightarrow \Omega_{c,1}[\Theta^n_{r,1}(P_2)] = 1$. Therefore, the examination phase is validated since proposition 6.5.1 is satisfied, and the investigator can proceed to the next phase, Analysis.

Analysis

Step1: The validation of analysis phase is shown in the following four phases:

Step1: Determine vectors **a** and **d**. The two AF techniques affecting the analysis phase are:

a_1 = Hash Collisions

a_2 = Loop References

d_1 = linear probing

d_2 = file name resolving error

Therefore, $a_1.d_1 = 1$ and $a_2.d_2 = 1$, since a_1 and a_2 are detected by d_1 and d_2. Step 2: Compute the detector matrix D_M when the values of the vectors **a** and **d** have been determined. By definition, $D_M = \begin{pmatrix} a_1.d_1 & a_2.d_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 \end{pmatrix}$. Therefore, $\Omega_{c,1}[\Theta^n_{r,1}(D_M)] = [1]_{1\times 1}$.

Step 3: Compute C_M once the AF techniques a_1 and a_2 have been detected, that is, DM has been computed. By definition, $C_M = a{*}C_M$. Now, **a** can be countered by $c_1 = c_2 = 1$, since it has been detected in step 2 (see special case in section). Therefore, $ap_1.c_1$ = logical 1 and $ap_2.c_2$ = logical 1. Therefore, $C_M = \begin{pmatrix} ap_1.c_1 & ap_2.c_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 \end{pmatrix}$. Therefore,

$\Omega_{c,1}[\Theta^n_{r,1}(C_M)] = {}_{c,1}\Phi^n_{c(r,1)}(P_3) = [1]_{1\times 1}$

Step 4: Compute phase function (P_f) once the values of D_M and C_M have been determined. From steps 2 and 3,, $P_{f_Analysis} = [1\ 1] \Rightarrow \Omega_{c,1}[\Theta^n_{r,1}(P_3)] = 1$. Therefore, the analysis phase is validated since proposition 6.5.1 is satisfied, and the investigator can proceed to the next phase, Reporting.

Reporting

Step1: The process of validating the reporting phase is shown in the following four steps:

Step1: First determine vectors **a** and **d**. The AF technique is affecting this phase is a_1 = Code Injection

$d_1 1$ = PsInfo Volatility Plugin

Therefore, $a_1.d_1 = 1$, since a_1 is detected by d_1.

Step 2: Compute D_M once **a** and **d** are determined. Therefore, $D_M = \begin{pmatrix} a_1.d_1 \end{pmatrix} = \begin{pmatrix} 1 \end{pmatrix}$. Therefore, $\Omega_{c,1}[\Theta^n_{r,1}(D_M)] = [1]_{1\times 1}$

Step 3: Compute C_M once the AF technique a_1 is detected, that is, D_M has been computed. Now, a_1 can be countered by c_1 = execute the software with elevated privileges. Therefore, $ap_1.c_1$ = logical 1, and by definition $C_M = \begin{pmatrix} a_1.c_1 \end{pmatrix} = \begin{pmatrix} 1 \end{pmatrix}$. Therefore, $\Omega_{c,1}[\Theta^n_{r,1}(C_M)] = [1]_{1\times 1}$

Step 4: Compute phase function (P_f) once the values of D_M and C_M are determined. Therefore, by definition, $P_{fReporting} = [1] \Rightarrow \Omega_{c,1}[\Theta^n_{r,1}(P_4)] = 1$. Therefore, the reporting phase is validated, and the investigator can stop here as reporting is the final phase of the DFM.

And, now since all the individual phase functions in the respective phases are logical 1. Mathematically, the P_f is an all ones unitary row vector i.e.,

$P_f = \begin{pmatrix} P_{fcollection} & P_{fexamination} & P_{fanalysis} & P_{freporting} \end{pmatrix}$. And since $P_f = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix} \Rightarrow {}_{r,1}\Phi^n_{c(r,1)}(P_n) = 1$. Therefore, by theorem 6.5.1 it can be concluded that this four-phase DFM is validated.

6.6.2 Invalidation Case

The DFMs Rekhis and Boudriga (2012b) and Rani and Kumari (2017) which consists of five and four phases respectively will also be evaluated because these are the two DFMs in the literature that consider to detect anti-forensic techniques in the DF process. For the Rekhis (Rekhis and Boudriga, 2012b) model, only the third phase function is $\Omega_{c,1}[\Theta^n_{r,1}(P_3)] = 1$, and the remaining phase functions of the individual phases are $\Omega_{c,1}[\Theta^n_{r,1}(P_1)] = \Omega_{c,1}[\Theta^n_{r,1}(P_2)] = \Omega_{c,1}[\Theta^n_{r,1}(P_4)] = \Omega_{c,1}[\Theta^n_{r,1}(P_5)] = 0$. This means the anti-forensic techniques are detected and countered only in the third phase, but the other phases are affected by anti-forensic attacks resulting in the overall phase function, ${}_{r,1}\Phi^n_{c(r,1)}(P_n) = 0$. Therefore by theorem 6.5.1(b), the model is invalid.

For the Rani's model Rani and Kumari (2017), the third phase function is also $\Omega_{c,1}[\Theta^n_{r,1}(P_3)] = 1$ the anti-forensic attacks are detected only in phase 3, but not during the other phases as the phase functions are $\Omega_{c,1}[\Theta^n_{r,1}(P_1)] = \Omega_{c,1}[\Theta^n_{r,1}(P_2)] = \Omega_{c,1}[\Theta^n_{r,1}(P_4)] = 0$ leading to the overall phase function $\Rightarrow {}_{r,1}\Phi^n_{c(r,1)}(P_n) = 0$. Therefore, according to theorem 6.5.1 (b), the model is invalid.

After evaluating the hypothetical model, and the other two models in the literature it has been found that the hypothetical model is valid for investigation purposes because it satisfies the validation principle. The other models only consider to detect anti-forensic attacks in the analysis phase, and not in other phases of the models. Therefore, by validation principle they are invalidated for investigation purposes.The advantage of this principle is, it just takes only four steps to validate a phase in the DFM. The principle systematically computes the AF techinques, detection methods, and countermeasure methods thereby facilitating a top-level view of the validation process which would be helpful to an investigator to validate or invalidate a DFM.

6.7 Conclusion

This chapter proposed a novel validation principle to validate Digital Forensic Models (DFMs) by answering RQ2, thereby addressing the research problem mentioned in 1.4. This principle can be used to assess or negate any risks prior to the investigation that arecaused by anti-forensic techniques. This will be useful to anyone making a validation plan or report according to FSR-G-218 which is under the umbrella of Forensic Science Regulator (Forensic Science Regulator, 2014) where in any risks must be assessed before proceeding to validate a method. Also, if the rebuttal presumption of correct working of computer rule is invoked by citing anti-forensics as the reason, then validation principle could be used to prove or disprove the reason for invocation of that rule, that is, whether that rule is applicable to a particular case or not. The validation principle falls on the higher layer of abstraction as seen in figure 6.1. The reason for this is because the validation principle can only validate a DFM if it has prior knowledge of an anti-forensic technique. It does not have the ability to predict a new anti-forensic technique such as zero-day attacks if it is not present in the database.

Chapter 7

Conclusion

7.1 Introduction

This chapter presents critical discussion between DNA fingerprinting method and the anti-forensic techniques identified in the wider context. Firstly, the contributions are summarized followed with how each of these contributions were made by answering the questions this research has posed. Secondly, the critical evaluation compares the functionality of DNA fingerprinting method with various memory acquisition techniques in the literature and highlights the advantages and shortcomings. Finally, the limitations of this research are presented along with suggested future research work directions from this study.

7.2 Summary of Contributions

The first contribution is that forensic memory acquisition can be detected on 64 bit-Windows using memory, I/O, and CPU features. This research proposed two methods to detect memory forensic acquistion. The first method was proposed by answering RQ1 in which various ML classifiers were analyzed for performance using the individual and integrated MIOC datasets respectively. After the analysis of the performance of ML classifiers on individual

MIOC datasets, it was found that if memory is aquired using memory and I/O parameters it outperforms when memory is being acquired by CPU parameters. Between memory and I/O, the I/O parameters perform well for forensic memory acquisition. It was found that by combining the MIOC parameters into a single dataset improved the performance of ML classifiers.

The second contribution is that forensic memory acquistion can be detected by distintive native attribute patterns using 3L-CNN. The second method proposed to detect forensic memory acquistion was to transform the samples in the individual and integrated MIOC datasets to images using the GSF property of the GAM. These images were then analysed using the 3L-CNN model. The individual and integrated datasets were subjected to multi-classification and binary classification. The individual multi-classification results show that the 3L-CNN overfits memory parametersand underfits the CPU parameters. The model fits I/O parameters data well. Therefore, it was concluded that when detecting forensic memory acquisition, I/O parameters exhibit good performance. For individual binary classification, the model again exibits high variance and overfits the memory parameters. As for I/O parameters the model fit the data well until epoch 37. The model's performance on CPU parameters was found to be satifactory.

The significance of the above contribution, in the context of forensic investigation is to create awareness to the forensic community that the forensic process could be vulnerable to the DNA fingerprint method. Specifically, during the evidence acquisition phase, vital or incriminating evidence could be stored in the system's physical memory which could be irrecoverably lost due to its volatile nature if it is not collected, unlike, for example, evidence that can be recovered when stored on a hard-drive. If a suspect implemented the DNA fingerprint method to detect memory acquisition, then they might have defeated the memory acquisition process, for example, by programmatically shutting down the machine when a forensic acquisition tool was run on the machine. Therefore, the investigator would

have lost physical memory evidence which could have been vital to a case.

The third contribution is a novel principle that can validate digital forensic models by counteracting anti-forensic techniques. A mathematical formalism was proposed by which every phase in the digital forensic process is to be validated by addressing or negating any anti-forensic techniques that influences a particular phase. Once, anti-forensic techniques in all phases of the digital forensic process have been addressed, then by the validation principle a digital forensic model can be validated. This study evaluates the principle with the help of two scenarions. First by showing a proof-of-concept scenario in which a DFM is validated. In the second scenario, two DFMs from the literature are selected as they consider anti-forensic effects into their forensic models. The evaluation using the validation principle has found that the models do not address anti-forensic techniques in each and every phase of the forensic process but only focus on the analysis phase to account for anti-forensic effects. The significance of the third contribution is that it will enable digital forensic investigators to take the anti-forensic angle into account whilst validating their tools and processes. This will create more confidence in their findings by ensuring validity through verification of their tools, techniques and procedures. Therefore, assuring the reliability of digital evidence which is one of the fundamental requirements of presenting digital evidence in the court of law.

7.3 Research Findings

This section presents the findings of the research and the extent to which each research question(s) were answered and their corresponding objective(s) met.

1. **RQ1**: How can forensic memory acquisition be accurately detected by artificial intelligence techniques?

   Objective 1: Differentiate between forensic memory acquisition and non acquisition by evaluating machine learning classifiers based on their accuracy, precision, and recall

Objective 2: Compare training-validation error curves with Bias-Variance tradeoff curves to determine if the model overfits, underfits, or fits the data well. Then, calculate the detection error rate for each ML classifier.

Eight supervised machine learning algorithms were used to classify between forensic memory acquisition and non-acquisition. The machine learning classifiers consists of memory, I/O, and CPU (MIOC) features individually in separate datasets. The dataset was split into train and validation sets and 5 k-fold cross validation was performed to estimate the model's skill on the data which is not present in the training set. The samples in the dataset formed inputs to the machine learning classifiers.

Whilst analysing the results memory features dataset, the RF classifier performs well when compared to other ML classifiers as it has precision, recall, and accuracy of 99.99 percent. To further support this evidence, its area under the ROC curve is also maximum, and its detection error rate is less than 5 percent which is due to the fact that it exhibits low bias and variance which means this model fits the data well.

The classifier that shows the least performance in the memory features dataset is the Ada Boost classifier, as it has an accuracy of 29.67 percent, and a precision and recall of 11.87 percent and 45.65 percent respectively. To further support these results, the area under its ROC curves show that classes 3 and 5 are effected, and this model suffers with high bias and variance with a detection error rate of 73.46 percent.

When the machine learning classifiers were learning features from the I/O dataset, once again the RF classifier performs well when compared to the other classifiers with precision, recall, and accuracy of 99.99 percent. This result is cross-verified with the area of the ROC curve which is maximum for this case. This model exhibits optimal bias and variance, and has a detection error rate of less than 5 percent, which means this model fits the data well. The model that underperforms on I/O dataset is, Ada Boost with accuracy of 51.5 percent and precision of 99 percent but recall of 24.56

percent. This model exhibits high bias with a detection error rate of 23.46 percent, and hence this model underfits the I/O dataset.

As for performance of the ML classifiers on the CPU dataset, all ML classifiers exhibit bias which means they underfit the CPU dataset. To address this problem the memory, I/O, and CPU datasets were combined to form an integrated dataset comprising of 20 features. In the combined feature dataset, the SVM and Ada Boost classifiers still exhibit high bias which mean they underfit the data. Whilst, other models exhibit low bias and variance and hence fit the data well.

Objective 5: Encode the samples in the dataset into images and classify them using convolutional neural network (CNN). Evaluate the model's perfomance by accuracy, precision, recall, false positives, and false negatives.

To extract the patterns from the dataset, the time-series data was encoded using the Gramian Summation Field property of the Gramian Angular Field. Each feature dataset, that is, memory, I/O, and CPU comprises of 14000 images in the training set and 3500 images in the test set. These images were classified using a 3-layered convolutional neural network (3L-CNN). To avoid overfitting, firstly the images were augmented by random transformations, so that it can be generalised better, and the model will not see the same image twice. Secondly, since the training dataset comprises of 14000 images only three convolutional layers are added in this model. Adding more than three layers, that is increasing the entropic capacity of the model, is not only time-cosuming but also will lead to overfitting as the model will even learn the noise patters from the datasets. Finally, a dropout layer is added after the activation layer, so that the model does not come across the same pattern twice. Therefore, data augmentation, entropic capacity, and dropout layer helps in reducing overfitting problems.

The 3L-CNN model was first analysed on individual MIOC datasets, and then combined to analyse the multi-classification, and binary classification results. The 3L-CNN

model has high recall but low precision, and overfits the memory features dataset. The performance of the model on the I/O dataset shows that it has a recall and precision of 1 and 0.99 respectively. The model fits the I/O dataset well and reduces overfitting to a greater extent. As for the CPU dataset the model suffers from low recall and precision. The model exhibits high variance and hence overfits the CPU dataset.

After multi-classification, binary classification was performed to improve the performance of the model on the MIOC features dataset. The model has overfit the memory feature data set as the model exhibits variance. The model performs well on the I/O feature dataset but it is recommened that this model must be stopped training at epoch number 37 and from epoch 38 the testing error curve remians constant but the training error curve starts to decrease. Therefore, the model starts to overfit the I/O feature dataset after epoch 38. As for the CPU dataset, overall the model fits the data well, but at epoch 20 an unusal instance is recorded. The model suffers from high bias which is unusal because due to the nature of the dataset and preventions taken against overfitting it is expected to see this model tackle overfitting and high variance, but not the opposite, that is, high bias and underfitting the data. The results show that this model overcomes underfitting after epoch 20 and continues to fit the data well.

After implementing individual classifications, integrated multi-classification and binary classification was performed by combining the features of memory, I/O, CPU into one single MIOC dataset. The integrated multi-classification the model underfits the data until epoch 35, and overcomes underfitting after epoch 35 and fits the data well. This is again unusal because the 3L-CNN model is designed to overcome overfitting and not underfitting. As for the integrated binary classification, the models fits the MIOC dataset well.

7.4 Critical Evaluation

The DNA fingerprinting, which is the proposed method in this research to detect forensic memory acquisition, will be compared to evaluate its functioning in the wider context, that is, to the existing methods in the literature that defeat forensic memory acquistion. In 2.3, this research has identified seven anti-forensic techniques (table 7.1) by which live memory acquisition can be defeated.

Direct kernel object manipulation (DKOM) is a technique in which the kernel objects are manipulated to facilitate changes in the operating system (Sparks and Butler, 2005) to the advantage of an attacker. For example, if the forward and backward link pointers of a process are manipulated, then a process can be hidden in the memory (Fuzen, 2015). During memory acquisition or live analysis this might not flag up as suspicious behaviour. When compared to the DNA fingerprinting method, the disadvantages of DKOM technique are that it can be unstable that could result in a blue screen of death (BSOD) (Fuzen, 2015). It is implemented as a kernel driver, which means in-depth understanding of the operating system is required. At times, this could mean working with undocumented kernel structures, and this is one of the reason that makes DKOM unstable. DKOM attack works only on 32-bit Windows OS, it does not work on 64-bit OS due to driver signing and kernel patch protection (KPP) (Microsoft, 2008) facility provided on 64-bit Windows OS.

AF Technique	Mode	Windows OS	ML Algorithm	Countermeasure
DKOM	Kernel	32-bit	No	KPP
Virtual Address Subversion	Kernel	32-bit	No	Driver Signing
Meterpreter	User	32-bit	No	Process handle modification
Ddefy	Kernel	32-bit	No	KPP
Dementia	Kernel	32-bit	No	Windows 8, 64-bit and above
Memory API Hooking	Kernel	32-bit	No	KPP
DECAF	User	32-bit	No	Process handle modification
DNA Fingerprinting	User	64-bit	Yes	Dynamic Analysis

Table 7.1 Comparison of DNA fingerprinting with AF techniques

Virtual memory subversion (VMS) technique has the ability to manipulate the content in memory, and also be undetected by anti-virus scanners (Sparks and Butler, 2005). A kernel driver is needed to facilitate VMS. This means it is an unstable method as it may cause BSOD, and may not be implemented on windows 64-bit machines due to KPP. Whereas, the DNA fingerprinting method is stable as it runs on 64-bit Windows OS in the user-mode as opposed to kernel mode.

The rootkit Ddefy (Bilby, 2006) defeats live forensic acquistion by incepting API calls both in user-mode and kernel-mode. In the user-mode, Ddefy performs dynamic link library (DLL) injection, and in kernel-mode it may adopt a different strategy such as service dispatcher table (SDT) hook or even perform a DKOM attack. The outcome of Ddefy is after live acquisition it produces a valid image with data hidden in it,

thus fooling the live acquisition tools. Ddefy functions on 32-bit Windows OS and only has the ability to acquire hard-disk contents when the system is in the on-state. Even though Ddefy and DNA fingerprinting technique are both live acquisition anti-forensic techniques, the difference in the functionality is that the latter acquires physical memory and the former acquires the contents of a hard-disk. DNA fingerprinting can be extended to detect if live hard-disk forensic is taking place, but it was not investigated during this study as it beyond the scope of this research.

Whilst Ddefy operates in the kernel and user mode to defeat live hard-disk forensic acquisition, the anti-forensic technique Dementia (Milković, 2012) also operates in the similar fashion to defeat live forensic memory acquisition even on 64-bit Windows OS up until Windows 7 but only in usermode. It has the ability to hide processes, threads, and other objects in the kernel. Dementia's anti-forensic techniques work on 32-bit Windows systems, and on 64-bit Windows only the user-mode method works. On the other hand, DNA fingerprinting technique works only in the user-mode on a Windows 10 machine, and does not adopt any invasive procedures such as DLL injection to defeat forensic memory acquisition.

Memory forensic acquisition tools at times use undocumented kernel APIs such as MmGetPhysical memor-ranges or MmMap-MemoryDump to acquire memory. These API calls can be intercepted to defeat memory acquisition (Stüttgen and Cohen, 2013). The shortcoming of this anti-forensic technique when compared to DNA fingerprinting technique is that it only functions on 32-bit Window OS.

The two anti-forensic tools that specifically operate in the user-mode are DECAF (Lim et al., 2012) and meterpreter (Kvitchko, 2015). These tools use signature based detection technique as demonstrated in 2.4 to defeat live memory acquisition. The shortcoming of this method is if the name of the forensic tool is changed, the signature based method will fail to function. This is where the DNA fingerprinting addresses this

shortcoming because it relies on the change in the MIOC-parameters to detect forensic memory acquisition rather than the name of the forensic tool. However, there are certain shortcomings of the DNA fingerprinting method. The DNA fingerprinting method was studied on Windows 10, 64-bit machines with 16 GB of memory. Therefore, arguably it may not function on differnt settings of Windows OS. The anti-forensic techniques that work in the kernel-mode have the ability to be stealthy; whereas, since the DNA fingerprint functions in the user-mode it does not implement measures to stay stealthy whilst it is detecting memory acquisition. This means if a memory image is acquired then it is likely that DNA fingerprinting method may flagged up using dynamic analysis (Sikorski and Honig, 2012, Pg 167). However, this is debatable as it adopts deep machine learning algorithm as seen in 5.4 to detect memory acquisition, and it does not appear that such AI techniques have been applied in domain of anti-forensics to detect live forensic memory acquisition as this study identified in 2.5.1.

7.5 Limitations of Study

The two memory acquisition procedures proposed in this research to detect forensic memory acquisition are supported on windows 10 64-bit machines. They may not be supported on other versions of Windows operating systems. The ML classifiers used the default parameters settings from the sklearn library. Due to time and resource constraints, hyperparameter tuning was not performed to determine the optimal values of parameters for the underformed ML classifiers. In Chapter 5, the size of the image was kept constant at 231x232 pixels. The 3L-CNN model was not tested too see how the performance of the model was affected due to various image sizes. This research only focused on detecting forensic memory acquisition, and does not focus on defeating the live acquisition process. However, the next step would be to create

a framework to defeat the live forensic acquisiton process which is informed by the methods used to detect forensic memeory acquistion proposed in this research.

7.6 Concluding Remarks

This research proposed two methods by which forensic memory acquisition can be detected. Firstly, it was hypothesized that memory acquisition can be detected if the variation pattern of the MIOC parameters can be recognised. Therefore, ML classifiers were used to detect the pattern variation, and it was found that Random Forest Classifier performed well with greater accuracy, precision and recall, and exhibited optimal bias and variance.

The other method proposed to detect forensic memory acquisition was to transform the MIOC samples to images using the GSF property of GAF. Then 3L-CNN was used to detect memory acquisition and non-acquisition patterns from these images with a greater accuracy, precision and recall.

Finally, a novel principle by which DFMs can be validated was proposed. During reviewing the literature it was found that various DFMS were being proposed by adding and deleting phases in the DF process. Instead of developing another DFM, this study proposes a mathematical formalisms to validate any DFM in the presence of AF attacks, and a paper was published in a reputed high impact factor journal in digital forensics.

7.7 Future Scope

The proposed methods of memory forensic detection is able to support the hypothesis in this research. However, further research has to investigate the challenges of detecting forensic memory acquisition in real-time. The following list contains several research ideas that can be pursued in the future as a continuation of the research presented in this thesis:

(a) The machine learning algorithms were implemented using sklearn ML libraries and only default parameters were enabled. One can perform the resource intensive method of hypertuning parameters to find the suitable parameters for the ML classifiers. This strategy may increase the accuracy, precision and recall for those ML classifiers that have underperformed in this research work.

(b) Only the GSF property of the GAF was used to transform MIOC feature samples to images. Other ways to transform MIOC features to images are GDF property of the GAF, and MTF to name a few. These transformations can be compared with each other to find out which transformation is most suited to the task of detecting forensic memory acquisition.

(c) The datasets produced in this research can be used as a baseline to be compared with other emerging datasets to detect forensic memory acquisiton.

(d) The python code can be turned to an executable file and then reverse engineered to ascertain how AI algorithms work in memory. This could open new doors in the field malware analysis-AI malware research.

(e) The mathematical principle proposed to validate DFMs by negating AF attacks can be implemented in a programming language and then evaluated as a proof-of-concept system.

References

Big data analytics solutions | mantech, 2015. URL https://www.mantech.com/capabilities/data-collection-and-analytics.

Belkasoft ram capturer: Volatile memory acquisition tool, 2019. URL https://belkasoft.com/ram-capturer.

Ftk imager version 4.2.0 | accessdata, 2019. URL https://accessdata.com/product-download/ftk-imager-version-4-2-0.

Magnet ram capture - magnet forensics, 2019. URL https://www.magnetforensics.com/resources/magnet-ram-capture/.

Releases - rekall forensics, 2019. URL http://www.rekall-forensic.com/releases.

Adaboost-bv-cpu, 2020a. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_CPU_Params/kfoldCV2/AdaBoost/.

Adaboost-bv-io, 2020b. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_IO_Params/kfoldCV2/AdaBoost/.

Adaboost-cm-cpu, 2020c. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_CPU_Params/kfoldCV2/AdaBoost/.

Adaboost-cm-io, 2020d. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_IO_Params/kfoldCV2/AdaBoost/.

Adaboost-pr-cpu, 2020e. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_CPU_Params/kfoldCV2/AdaBoost/.

Adaboost-pr-io, 2020f. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_IO_Params/kfoldCV2/AdaBoost/.

Adaboost-roc-cpu, 2020g. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_CPU_Params/kfoldCV2/AdaBoost/.

Adaboost-roc-io, 2020h. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_IO_Params/kfoldCV2/AdaBoost/.

Gaussiannb-bv-cpu, 2020a. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_CPU_Params/kfoldCV2/GaussianNB/.

Gaussiannb-cm-cpu, 2020b. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_CPU_Params/kfoldCV2/GaussianNB/.

Gaussiannb-pr-cpu, 2020c. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_CPU_Params/kfoldCV2/GaussianNB/.

Gaussiannb-roc-mem, 2020d. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_Mem_Params/kfoldCV2/GaussianNB/.

Gaussiannb-roc-cpu, 2020e. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_CPU_Params/kfoldCV2/GaussianNB/.

Lda-bv-cpu, 2020a. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_CPU_Params/kfoldCV2/LDA/.

Lda-bv-io, 2020b. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_IO_Params/kfoldCV2/LDA/.

Lda-cm-cpu, 2020c. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_CPU_Params/kfoldCV2/LDA/.

Lda-cm-io, 2020d. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_IO_Params/kfoldCV2/LDA/.

Lda-pr-cpu, 2020e. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_CPU_Params/kfoldCV2/LDA/.

Lda-pr-io, 2020f. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_IO_Params/kfoldCV2/LDA/.

Lda-roc-cpu, 2020g. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_CPU_Params/kfoldCV2/LDA/.

Lda-roc-io, 2020h. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_IO_Params/kfoldCV2/LDA/.

Memadaboostbv, 2020a. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_Mem_Params/kfoldCV2/AdaBoost/.

Memadaboostcm, 2020b. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_Mem_Params/kfoldCV2/AdaBoost/.

Memadaboostpr, 2020c. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_Mem_Params/kfoldCV2/AdaBoost/.

Memadaboostroc, 2020d. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_Mem_Params/kfoldCV2/AdaBoost/.

pandas: powerful python data analysis toolkit, 2020. URL https://pandas.pydata.org/docs/pandas.pdf.

Svm-bv-io, 2020a. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_IO_Params/kfoldCV2/SVM/.

Svm-cm-io, 2020b. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_IO_Params/kfoldCV2/SVM/.

Svm-pr-io, 2020c. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_IO_Params/kfoldCV2/SVM/.

Svm-roc-io, 2020d. URL https://bitbucket.org/PhD_DF/data-analysis/src/master/Machine%20Learning/Dataset/Dataset_IO_Params/kfoldCV2/SVM/.

Nurul Hidayah Ab Rahman, William Bradley Glisson, Yanjiang Yang, and Kim Kwang Raymond Choo. Forensic-by-Design Framework for Cyber-Physical Cloud Systems. *IEEE Cloud Computing*, 3(1):50–59, 2016. ISSN 23256095. doi: 10.1109/MCC.2016.5.

Frank Adelstein. Live forensics: diagnosing your system without killing it first. *Communications of the ACM*, 49(2):63–66, 2006.

Ankit Agarwal, Megha Gupta, Saurabh Gupta, and Subhash Chandra Gupta. Systematic Digital Forensic Investigation Model. *Gupta International Journal of Computer Science and Security*, 2011. ISSN 19385862. doi: 10.1149/1.2992231.

Waqas Ahmed and Baber Aslam. A comparison of windows physical memory acquisition tools. In *MILCOM 2015-2015 IEEE Military Communications Conference*, pages 1292–1297. IEEE, 2015.

Mohammad Alauthman. *a Reinforcement Learning Technique M M Alauthman*. 2016.

Abdulalem Ali, Shukor Abd Razak, Siti Hajar Othman, Arafat Mohammed, and Faisal Saeed. A metamodel for mobile forensics investigation domain. *PLoS ONE*, 2017. ISSN 19326203. doi: 10.1371/journal.pone.0176223.

Srikar Appalaraju and Vineet Chaoji. Image similarity using Deep CNN and Curriculum Learning. pages 1–9, 2017. URL http://arxiv.org/abs/1709.08761.

Harald Baier and Julian Knauer. AFAUC-Anti-forensics of storage devices by alternative use of communication channels. In *Proceedings - 8th International Conference on IT Security Incident Management and IT Forensics, IMF 2014*, 2014. ISBN 9781479943302. doi: 10.1109/IMF.2014.11.

Fernando Gertum Becker. *The essential guide to effect sizes: statistical power, meta-analysis, and the interpretation of research results*, volume 48. 2011. ISBN 9780521142465. doi: 10.5860/choice.48-5742.

Darren Bilby. Low down and dirty: Anti-forensic rootkits, 2006. URL https://www.blackhat.com/presentations/bh-jp-06/BH-JP-06-Bilby-up.pdf.

Bitbucket. $\mathrm{Phd}_d f/data-analysis/machinelearning/dataset/dataset_c pu_p arams/dataset_c puparams.csv^{-}$

Bitbucket. $\mathrm{Phd}_d f/data-analysis/machinelearning/dataset/dataset_i o_p arams/dataset_i oparams.csv^{-}bitbuck$

Bitbucket. $\mathrm{Phd}_d f/data-analysis/machinelearning/dataset/combined_d ataset^{-}bitbucket, 2020c.URL.$

Bitbucket. $\mathrm{Phd}_d f/data-analysis/machinelearning/dataset/dataset_m em_p arams/dataset_m emparams.csv—b$

Carrier. Open Source Digital Forensics Tools : The Legal Argument. *October*, 2002.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 13-17-August-2016:785–794, 2016. 10.1145/2939672.2939785.

Yan Ju Chen, Wen Han Kuo, Sung Yun Tsai, Jiann Liang Chen, Yu Hung Chen, and Wei Zhao Xu. Artificial Intelligence Hybrid Learning Architecture for Malware Families Classification. *International Conference on Advanced Communication Technology, ICACT*, 2019-February: 503–510, 2019. ISSN 17389445. 10.23919/ICACT.2019.8701899.

Zhengyang Chen, Bowen Yu, Yu Zhang, Jianzhong Zhang, and Jingdong Xu. Automatic mobile application traffic identification by convolutional neural networks. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 301–307. IEEE, 2016.

F. Mulier Cherkassy. *Learning from Data - Concepts, Theory and Methods 2nd ed (Wiley, 2007)*. ISBN 9780471681823.

S Ciardhuáin, Nicole Lang Beebe, Jan Guynes Clark, Gary Palmer, Sundresan Perumal, Siti Rahayu Selamat, Robiah Yusof, and Shahrin Sahib. An extended model of cybercrime investigations. *the First Digital Forensic Research Workshop (DFRWS)*, 2009. ISSN 1751-911X. 10.1504/IJESDF.2010.033780.

Francois Collet. Updated to the keras 2.0 api. · github, 2016. URL https://gist.github.com/fchollet/0830affa1f7f19fd47b06d4cf89ed44d.

Courts and Tribunals Judiciary. Criminal practice directions, 2014. URL https://www.judiciary.uk/wp-content/uploads/JCO/Documents/Practice+Directions/Consolidated-criminal/criminal-practice-directions-2013.pdf.

Kamal Dahbur and Bassil Mohammad. The anti-forensics challenge. In *Proceedings of the 2011 International Conference on Intelligent Semantic Web-Services and Applications - ISWSA '11*, 2011. ISBN 9781450304740. 10.1145/1980822.1980836.

Yusheng Dai, Hui Li, Yekui Qian, and Xidong Lu. A malware classification method based on memory dump grayscale image. *Digital Investigation*, 27:30–37, 2018. ISSN 17422876. 10.1016/j.diin.2018.09.006. URL https://doi.org/10.1016/j.diin.2018.09.006.

Pınar Dönmez. Introduction to machine learning, by ethem alpaydın. cambridge, ma: The mit press2010. isbn: 978-0-262-01243-0. $54/£39.95+584 pages. Natural\ Language\ Engineering, 19(2), 2013.$

D A Duce, F R Mitchell, P Turner, Madjid Merabti, and Computer Security. article : the Use of Artificial Intelligence in Digital Forensics : an. 7:35–41, 2010.

Forensic Science Regulator. Fsr guidance validation, 2014. URL https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/375285/FSR-G-201_Validation_guidance_November_2014.pdf.

Forensic Science Regulator. Method validation in digital forensics, 2016. URL https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/528123/FSR_Method_Validation_in_Digital_Forensics_FSR-G-218_Issue_1.pdf.

Forensic Science Regulator. Forensic science regulator, 2019. URL https://www.gov.uk/government/organisations/forensic-science-regulator/about/membership#forensic-science-advisory-council.

Fuzen. rootkit.com/fu$_r eadme.txt at master$ů$bowl of stew/rootkit.com$ů$github, 2015. URL.

Simson Garfinkel. Anti-Forensics : Techniques , Detection and Countermeasures. *Security*, 2006. ISSN 0091-679X. 10.1.1.109.5063.

Vera Goebel and Thomas Plagemann. Research / Scientific Methods in Computer Science What is Science ? (6), 2009.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

David E Gray. Theoretical perspectives and research methodologies. *Doing research in the real world*, pages 16–38, 2014. URL http://www.uk.sagepub.com/books/Book239646{#}tabview=toc.

Michael Gruhn and Felix C Freiling. Evaluating atomicity, and integrity of correct memory acquisition methods. *Digital Investigation*, 16:S1–S10, 2016.

Murat Gül and Emin Kugu. A survey on anti-forensics techniques. In *Artificial Intelligence and Data Processing Symposium (IDAP), 2017 International*, pages 1–6. IEEE, 2017.

Malek Harbawi and Asaf Varol. An improved digital evidence acquisition model for the Internet of Things forensic I: A theoretical framework. *2017 5th International Symposium on Digital Forensic and Security, ISDFS 2017*, pages 1–6, 2017. 10.1109/ISDFS.2017.7916508.

Ryan Harris. Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem. *Digital Investigation*, 2006. ISSN 17422876. 10.1016/j.diin.2006.06.005.

Takahiro Haruyama and Hiroshi Suzuki. One-byte modification for breaking memory forensic analysis. *Black Hat Europe*, 2012.

Brian Hay, Matt Bishop, and Kara Nance. Live analysis: Progress and challenges. *IEEE Security & Privacy*, 7(2):30–37, 2009.

Roberta Heale and Dorothy Forbes. Understanding triangulation in research. *Evidence-Based Nursing*, 16(4):98, 2013. ISSN 13676539. 10.1136/eb-2013-101494.

Henry. Quantitative methods in the triangulation process, 2018. URL https://delaat.net/rp/2014-2015/p50/presentation.pdf.

Ben Hitchcock, Nhien-An Le-Khac, and Mark Scanlon. Tiered forensic methodology model for digital field triage by non-digital evidence specialists. *Digital investigation*, 16:S75–S85, 2016.

M Hossin and MN Sulaiman. A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process*, 5 (2):01–11, 2015. ISSN 2231007X. 10.5121/ijdkp.2015.5201.

House of Commons Science Technology Committee. Forensic science on trial, 2019. URL https://publications.parliament.uk/pa/cm200405/cmselect/cmsctech/96/96i.pdf.

International Standards Organisation. Iso 17025:2017 general requirements for the competence of testing and calibration laboratories, 2017. URL https://www.iso.org/standard/66912.html.

Mike Jacobs and Michael Satran. Detecting media insertion or removal - win32 apps | microsoft docs, 2017a. URL https://docs.microsoft.com/en-us/windows/win32/devio/detecting-media-insertion-or-removal.

Mike Jacobs and Michael Satran. How to shut down the system - win32 apps | microsoft docs, 2017b. URL https://docs.microsoft.com/en-us/windows/win32/shutdown/how-to-shut-down-the-system.

Brownlee Jason. *Mastering Machine Learning Algorithms*. 2016.

Jin-Su Jeong and Yong-Ju Kwon. Definition of Scientific Hypothesis: A Generalization or a Causal Explanation?, 2006. ISSN 1226-5187.

Grigorios Kalliatakis, Shoaib Ehsan, Ales Leonardis, Maria Fasli, and Klaus D. McDonald-Maier. Exploring object-centric and scene-centric CNN features and their complementarity for human rights violations recognition in images. *IEEE Access*, 7:10045–10056, 2019. ISSN 21693536. 10.1109/ACCESS.2019.2891745.

Da Yu Kao and Guan Jie Wu. A Digital Triage Forensics framework of Window malware forensic toolkit: Based on ISO/IEC 27037:2012. *Proceedings - International Carnahan Conference on Security Technology*, 2015-January:217–222, 2016. ISSN 10716572. 10.1109/CCST.2015.7389685.

Nickson M Karie, Victor R Kebande, HS Venter, and Kim-Kwang Raymond Choo. On the importance of standardising the process of generating digital forensic reports. *Forensic Science International: Reports*, 1:100008, 2019.

Prabhjot Kaur, Anchit Bijalwan, RC Joshi, and Amit Awasthi. Network forensic process model and framework: An alternative scenario. In *Intelligent Communication, Control and Devices*, pages 493–502. Springer, 2018.

Victor R. Kebande, Nickson M. Karie, Antonia Michael, Semaka Malapane, Ivans Kigwana, H. S. Venter, and Ruth D. Wario. Towards an integrated digital forensic investigation framework for an IoT-based ecosystem. *Proceedings - 2018 IEEE International Conference on Smart Internet of Things, SmartIoT 2018*, pages 93–98, 2018. 10.1109/SmartIoT.2018.00-19.

Ivans Kigwana, Victor R. Kebande, and H. S. Venter. A proposed digital forensic investigation framework for an eGovernment structure for Uganda. *2017 IST-Africa Week Conference, IST-Africa 2017*, pages 1–8, 2017. 10.23919/ISTAFRICA.2017.8102348.

Kvitchko. Metasploit cheatsheet, 2015. URL https://www.sans.org/security-resources/sec560/misc_tools_sheet_v1.pdf.

Frank YW Law, Pierre KY Lai, KP Chow, Ricci SC Ieong, Michael YK Kwan, Kenneth WH Tse, and KS Hayson. Memory acquisition: A 2-take approach. In *Proceedings of the International Conference on Computer Science and Its Applications, CSA 2009. IEEE.*, 2009.

Quan Le, Oisín Boydell, Brian Mac Namee, and Mark Scanlon. Deep learning at the shallow end: Malware classification for non-domain experts. *Proceedings of the Digital Forensic Research Conference, DFRWS 2018 USA*, 26:S118–S126, 2018. ISSN 17422876. 10.1016/j.diin.2018.04.024. URL https://doi.org/10.1016/j.diin.2018.04.024.

Fei Li. Buzzword : CNN. 2018.

Kyung-Soo Lim, Antonio Savoldi, Changhoon Lee, and Sangjin Lee. On-the-spot digital investigation by means of ldfs: Live data forensic system. *Mathematical and computer modelling*, 55(1-2):223–240, 2012.

Raymond Lutui. A multidisciplinary digital forensic investigation process model. *Business Horizons*, 59(6):593–604, 2016.

Aziz Makandar and Anita Patrot. Malware class recognition using image processing techniques. *2017 International Conference on Data Management, Analytics and Innovation, ICDMAI 2017*, pages 76–80, 2017. 10.1109/ICDMAI.2017.8073489.

Steve Mansfield-Devine. Fighting forensics. *Computer Fraud & Security*, 2010(1):17–20, 2010.

Zane Markel and Michael Bilzor. Building a machine learning classifier for malware detection. *WATeR 2014 - Proceedings of the 2014 2nd Workshop on Anti-Malware Testing Research*, pages 1–4, 2015. 10.1109/WATeR.2014.7015757.

Angus M. Marshall and Richard Paige. Requirements in digital forensics method definition: Observations from a UK study. *Digital Investigation*, 27:23–29, 2018. ISSN 17422876. 10.1016/j.diin.2018.09.004.

MASH, 2005. URL http://www.mash.dept.shef.ac.uk/Resources/inversefunctions.pdf.

Robert J McDown, Cihan Varol, Leonardo Carvajal, and Lei Chen. In-depth analysis of computer memory acquisition software for forensic purposes. *Journal of forensic sciences*, 61:S110–S116, 2016.

Matthew Meyers and Marcus Rogers. Digital forensics: Meeting the challenges of scientific evidence. In *IFIP International Conference on Digital Forensics*, pages 43–50. Springer, 2005.

Microsoft. Microsoft security advisory 932596 | microsoft docs, 2008. URL https://docs.microsoft.com/en-us/security-updates/securityadvisories/2007/932596.

Luka Milković. Defeating Windows memory forensics. *29C3*, 2012.

Reza Montasari. Review and Assessment of the Existing Digital Forensic Investigation Process Models. *International Journal of Computer Applications*, 2016. ISSN 09758887. 10.5120/ijca2016911194.

Sasa Mrdovic, Alvin Huseinovic, and Ernedin Zajko. Combining static and live digital forensic analysis in virtual environment. In *2009 XXII International Symposium on Information, Communication and Automation Technologies*, pages 1–6. IEEE, 2009.

National Institute of Standards and Technology. Computer forensics tool testing program. URL https://www.nist.gov/itl/ssd/software-quality-group/computer-forensics-tool-testing-program-cftt.

Ori Or-Meir, Nir Nissim, Yuval Elovici, and Lior Rokach. Dynamic malware analysis in the modern era—a state of the art survey. *ACM Computing Surveys (CSUR)*, 52(5):1–48, 2019.

Edewede Oriwoh, David Jazani, Gregory Epiphaniou, and Paul Sant. Internet of Things Forensics: Challenges and Approaches. *9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 608–615, 2013. 10.4108/icst.collaboratecom.2013.254159.

Darren Quick and Kim-Kwang Raymond Choo. Impacts of increasing volume of digital forensic data: A survey and future research challenges. *Digital Investigation*, 11(4):273–294, 2014.

Deevi Radha Rani and G. Geetha Kumari. A framework for detecting anti-forensics in cloud environment. *Proceeding - IEEE International Conference on Computing, Communication and Automation, ICCCA 2016*, pages 1277–1280, 2017. 10.1109/CCAA.2016.7813913.

Slim Rekhis and Noureddine Boudriga. A system for formal digital forensic investigation aware of anti-forensic attacks. *IEEE Transactions on Information Forensics and Security*, 7 (2):635–650, 2012a. ISSN 15566013. 10.1109/TIFS.2011.2176117.

Slim Rekhis, S and Noureddine Boudriga. A Hierarchical Visibility theory for formal digital investigation of anti-forensic attacks. *Computers and Security*, 31(8):967–982, 2012b. ISSN 01674048. 10.1016/j.cose.2012.06.009. URL http://dx.doi.org/10.1016/j.cose.2012.06.009.

A Reyes, R Brittson, K O'Shea, and J Steele. Incident response: Live forensics and investigations. *Cyber Crime Investigations, Elsevier*, pages 89–109, 2007.

Giampaolo Rodola. psutil · pypi, 2019. URL https://pypi.org/project/psutil/.

Marcus Rogers, James Goldman, Rick Mislan, Timothy Wedge, and Steve Debrota. Computer Forensics Field Triage Process Model. *The Journal of Digital Forensics, Security and Law*, 2006. ISSN 15587223. 10.15394/jdfsl.2006.1004.

Chris Rowen. Using convolutional neural networks for image recognition, 2015. URL https://ip.cadence.com/uploads/901/cnn_wp-pdf.

Stuart Russell and Peter Norvig. *Artificial Intelligence A Modern Approach Third Edition*. 2010. ISBN 9780136042594. 10.1017/S0269888900007724.

Mark Russinovich, David Solomon, and Alex Ionescu. *Windows Internals, Part 1 (6th Edition)*. 2012a. ISBN 978-0735648379. URL http://technet.microsoft.com/en-gb/sysinternals/bb963901.aspx.

Mark Russinovich, David Solomon, and Alex Ionescu. *Windows Internals, Part 2 (6th Edition)*. 2012b. ISBN 978-0735665873. URL http://technet.microsoft.com/en-gb/sysinternals/bb963901.aspx.

Scientific Working Group on Digital Evidence. Model standard operation procedures for computer forensics, 2019. URL https://www.swgde.org/documents/Current%20Documents/SWGDE%20QAM%20and%20SOP%20Manuals/SWGDE%20Model%20SOP%20for%20Computer%20Forensics.

Mohit Sewak, Sanjay K. Sahay, and Hemant Rathore. Comparison of deep learning and the classical machine learning algorithm for the malware detection. *Proceedings - 2018 IEEE/ACIS 19th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2018*, pages 293–296, 2018. 10.1109/SNPD.2018.8441123.

Syed Zainudeen Mohd Shaid and Mohd Aizaini Maarof. In memory detection of windows api call hooking technique. In *2015 international conference on computer, communications, and control technology (i4CT)*, pages 294–298. IEEE, 2015.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*, volume 9781107057135. 2013. ISBN 9781107298019. 10.1017/CBO9781107298019.

Rami Sihwail, Khairuddin Omar, Khairul Akram Zainol Ariffin, and Sanad Al Afghani. Malware detection approach based on artifacts in memory image and dynamic analysis. *Applied Sciences (Switzerland)*, 9(18), 2019. ISSN 20763417. 10.3390/app9183680.

Michael Sikorski and Andrew Honig. *Practical malware analysis: the hands-on guide to dissecting malicious software. no starch press*. 2012. ISBN 9788578110796. 10.1017/CBO9781107415324.004.

sklearnCM. sklearn.metrics.confusion$_matrix$—*scikit* − *learn*0.22.1*documentation*, 2019.*URL*

Somayeh Soltani and Seyed Amin Hosseini Seno. A formal model for event reconstruction in digital forensic investigation. *Digital Investigation*, 2019.

Sherri Sparks and Jamie Butler. Shadow Walker - Raising the Bar for Rootkint Detection. *Black Hat Japan*, 2005. URL http://blackhat.com/presentations/bh-usa-05/bh-us-05-sparks.pdf.

Statmodels. statsmodels.stats.power.ttestindpower — statsmodels, 2019. URL http://www.statsmodels.org/dev/generated/statsmodels.stats.power.TTestIndPower.html.

Marsland Stephen. *Machine Learning An Algorithmic Perspective Second Edition*. 2014. ISBN 9781420067187.

Johannes Stüttgen and Michael Cohen. Anti-forensic resilient memory acquisition. *Digital investigation*, 10:S105–S115, 2013.

M Suiche. Win32dd, 2009, 2013.

Tanki. Tanki online wiki - tanki online wiki, 2018. URL https://en.tankiwiki.com/Main_Page.

The National Archives. Police and criminal evidence act 1984, 2019. URL http://www.legislation.gov.uk/ukpga/1984/60/contents.

The National Archives. Youth justice and criminal evidence act 1999, 2019. URL http://www.legislation.gov.uk/ukpga/1999/23/contents.

Yun-Cheng Tsai, Jun-Hao Chen, and Chun-Chieh Wang. Encoding Candlesticks as Images for Patterns Classification Using Convolutional Neural Networks. pages 1–22, 2019. URL http://arxiv.org/abs/1901.05237.

Upcounsel. The federal standard of expert testimony reliability before daubert, 2019. URL https://tinyurl.com/yxa5gpdx.

R Vinayakumar, KP Soman, and Prabaharan Poornachandran. Applying convolutional neural network for network intrusion detection. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1222–1228. IEEE, 2017.

R. Vinayakumar, Mamoun Alazab, K. P. Soman, Prabaharan Poornachandran, and Sitalakshmi Venkatraman. Robust Intelligent Malware Detection Using Deep Learning. *IEEE Access*, 7:46717–46738, 2019. ISSN 21693536. 10.1109/ACCESS.2019.2906934.

Stefan Vömel and Felix C Freiling. Correctness, atomicity, and integrity: defining criteria for forensically-sound memory acquisition. *Digital Investigation*, 9(2):125–137, 2012.

Stefan Vömel and Johannes Stüttgen. An evaluation platform for forensic memory acquisition software. *Digital Investigation*, 10:S30–S40, 2013.

Aaron Walters and Nick L Petroni. Volatools: Integrating volatile memory into the digital investigation process. *Black Hat DC*, 2007:1–18, 2007.

Zhiguang Wang and Tim Oates. Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. *AAAI Workshop - Technical Report*, WS-15-14:40–46, 2015.

Alan Weir. Formalism in the philosophy of mathematics. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, 2019.

Williams. Acpo good practice guide acpo good practice guide for digital evidence, 2012. URL http://library.college.police.uk/docs/acpo/digital-evidence-2012.pdf.

Martin Wundram, Felix C. Freiling, and Christian Moch. Anti-forensics: The next step in digital forensics tool testing. In *Proceedings - 7th International Conference on IT Security Incident Management and IT Forensics, IMF 2013*, 2013. ISBN 9780769549552. 10.1109/IMF.2013.17.

Yang Xin, Lingshuang Kong, Zhi Liu, Yuling Chen, Yanmiao Li, Hongliang Zhu, Mingcheng Gao, Haixia Hou, and Chunhua Wang. Machine learning and deep learning methods for cybersecurity. *IEEE Access*, 6:35365–35381, 2018.

Jace. Xu. Pyautoit · pypi, 2016. URL https://pypi.org/project/PyAutoIt/.

Yunus Yusoff, Roslan Ismail, and Zainuddin Hassan. Common Phases of Computer Forensics Investigation Models. *International Journal of Computer Science and Information Technology*, 2011. ISSN 09754660. 10.5121/ijcsit.2011.3302.

Ning Zhang, Ruide Zhang, Kun Sun, Wenjing Lou, Y Thomas Hou, and Sushil Jajodia. Memory forensic challenges under misused architectural features. *IEEE Transactions on Information Forensics and Security*, 13(9):2345–2358, 2018.

Tanveer Zia, Peng Liu, and Weili Han. Application-Specific Digital Forensics Investigative Model in Internet of Things (IoT). In *Proceedings of the 12th International Conference on Availability, Reliability and Security - ARES '17*, 2017. ISBN 9781450352574. 10.1145/3098954.3104052.

Appendix A

SVM Curves

A.1 Learning Curves for Intergrated MIOC

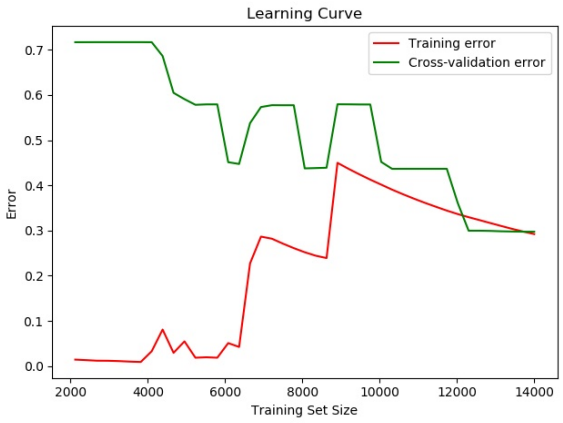


Fig. A.1 SVM Learning Curve

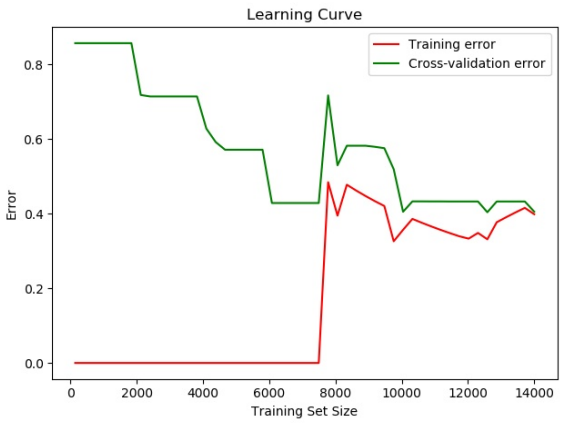


Fig. A.2 ADA Bosst Learning Curve

Appendix B

Memory Parameters

B.1 Memory Parameters

Memory Parameter	Description
Non-paged Pool	Amount of memory a process is using that cannot be moved out to the pagefile and remains in memory.
Numpage	The number of pagefaults
Paged_Pool	Amount of memory a process is using in pageable memory region.
Pagefile	The data that is not stored in the physical memory
Peak_nonpaged_pool	Maximum value of nonpaged pool in bytes
Peak_paged_pool	Maximum value of paged pool in bytes
Peak_pagefile	Maximum value of pagefile
Peak_wset	Peak value of Wset
Unique set size (USS)	Memory that is freed when a process terminates. Every process has its own USS.
Wset	The non-swapped memory a process has utilised. It is also known as resident set size.

Table B.1 Memory Parameters

Appendix C

I/O Parameters

C.1 I/O Parameters

I/O Parameters	Description
Read_count	Number of read operations performed
Write_count	Number of write operations performed
Read_bytes	Number of bytes read
Write_bytes	Number of bytes written
Other_count	Number of operations other than read and write
Other_bytes	Number of bytes transferred other than read and write operations

Table C.1 IO Parameters

C.2 CPU Parameters

CPU Parameters	Description
Kernel	Amount of time CPU spends in kernel-mode
User	Amount of time CPU spends in user-mode.
Blocking	The time that is elapsed before and after the interval
Non-blocking	The time elapsed since last function call or module import

Table C.2 CPU Parameters

Appendix D

Memory Parameters Evaluation Metrics

D.1 Multiclassification Memory Metrics

Metric	Training	Validation
Accuracy	96.14	85.71
Loss	10.33	23.47
Precision	82.02	50.25
Recall	97.60	100

Table D.1 Memory Multiclassification Evaluation Metrics

D.2 Binary Classification Memory Metrics

Metric	Training	Validation
Accuracy	94.14	86.23
Loss	12.33	23.66
Precision	76.98	57.21
Recall	94.80	82.23
FP	150	62
FN	700	500

Table D.2 Memory Binary-classification Evaluation Metrics

D.3 I/O Multiclassification Evaluation Metrics

Metric	Training	Validation
Accuracy	99.29	99.77
Loss	3.04	2.86
Precision	99.80	100
Recall	99.75	99.20

Table D.3 IO Multiclassification Evaluation Metrics

D.4 I/O Binary Classification Evaluation Metrics

Metric	Training	Validation
Accuracy	99.97	99.89
Loss	0.46	1.82
Precision	99.95	99.80
Recall	100	100
FP	4	4
FN	0	0

Table D.4 I/O Binary-classification Evaluation Metrics

D.5 CPU Parameters Multiclassification Metrics

Metric	Training	Validation
Accuracy	68.77	64.94
Loss	67.47	91.91
Precision	71.18	69.50
Recall	42.85	44.20

Table D.5 CPU Multiclassification Evaluation Metrics

D.6 CPU Parameters Binary Classification Metrics

Metric	Training	Validation
Accuracy	94.90	96.37
Loss	16.64	11.80
Precision	94.51	95.57
Recall	96.69	98.20
False Positives	449	91
False Negatives	265	36

Table D.6 CPU Binary Classification Metrics

D.7 Integrated Multiclassification Metrics

Metric	Training	Validation
Accuracy	97.09	97.71
Loss	7.32	5.80
Precision	99.95	100
Recall	99.85	99.20

Table D.7 Integrated Multiclassification Metrics

D.8 Integrated Binary Classification Metrics

Metric	Training	Validation
Accuracy	94.90	96.37
Loss	16.64	11.80
Precision	94.51	95.57
Recall	96.69	98.20
False Positives	449	91
False Negatives	265	36

Table D.8 CPU Binary Classification Metrics